Paceline

Improving Single-Thread Performance through Core Overclocking

Brian Greskamp and Josep Torrellas
http://iacoma.cs.uiuc.edu/

University of Illinois
at Urbana-Champaign

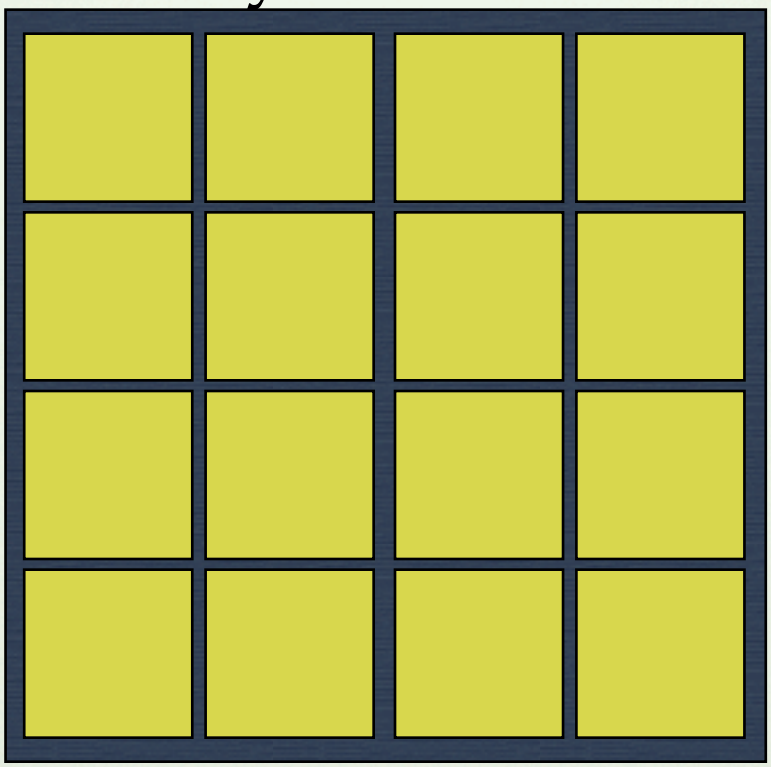
i-acoma group

Motivation

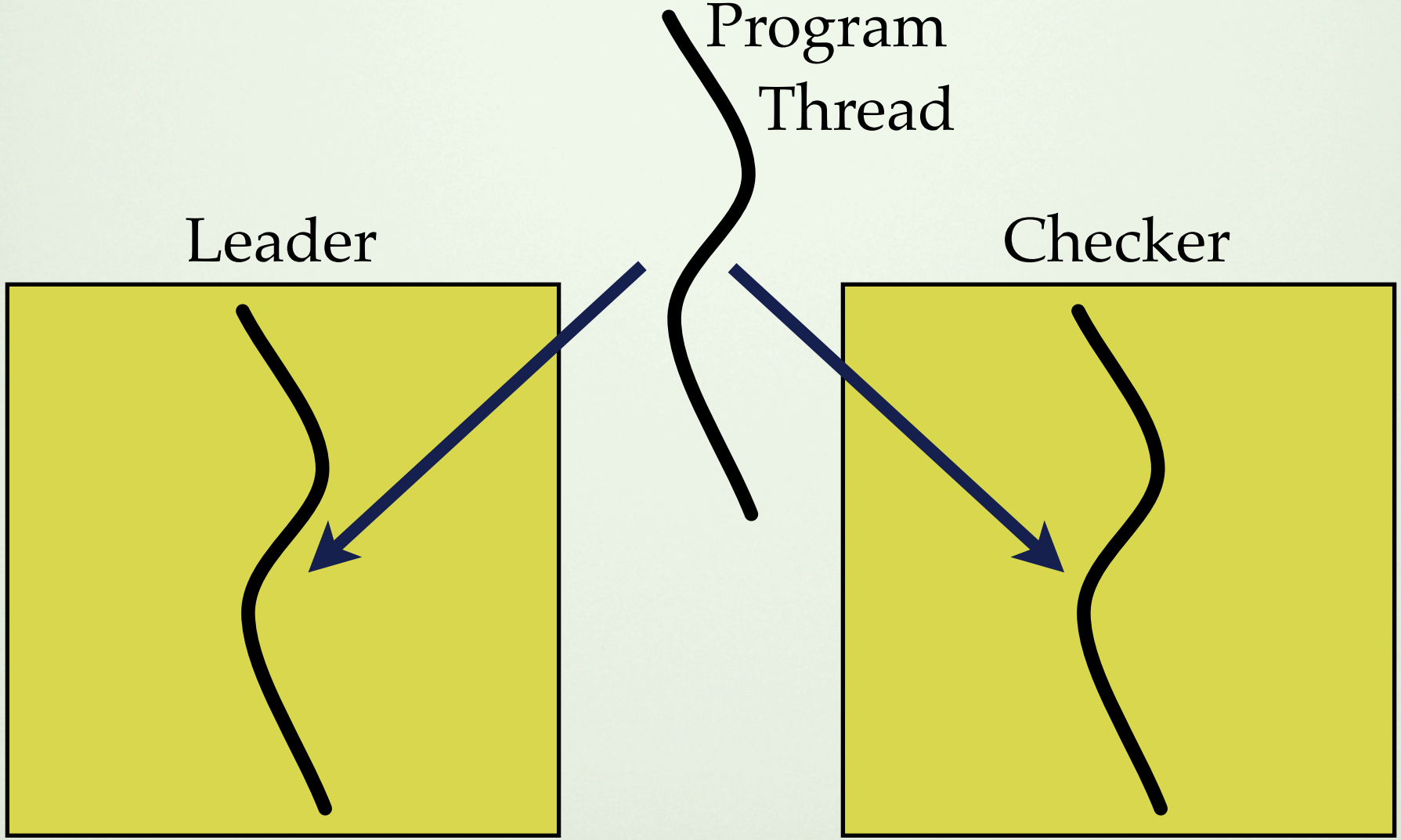
- Frequency growth is slowing
- CMP trend
  - Moderate IPC per core
  - Plenty of cores to spare
- How to improve single-thread performance?

Paceline Concept

Many-Core CMP



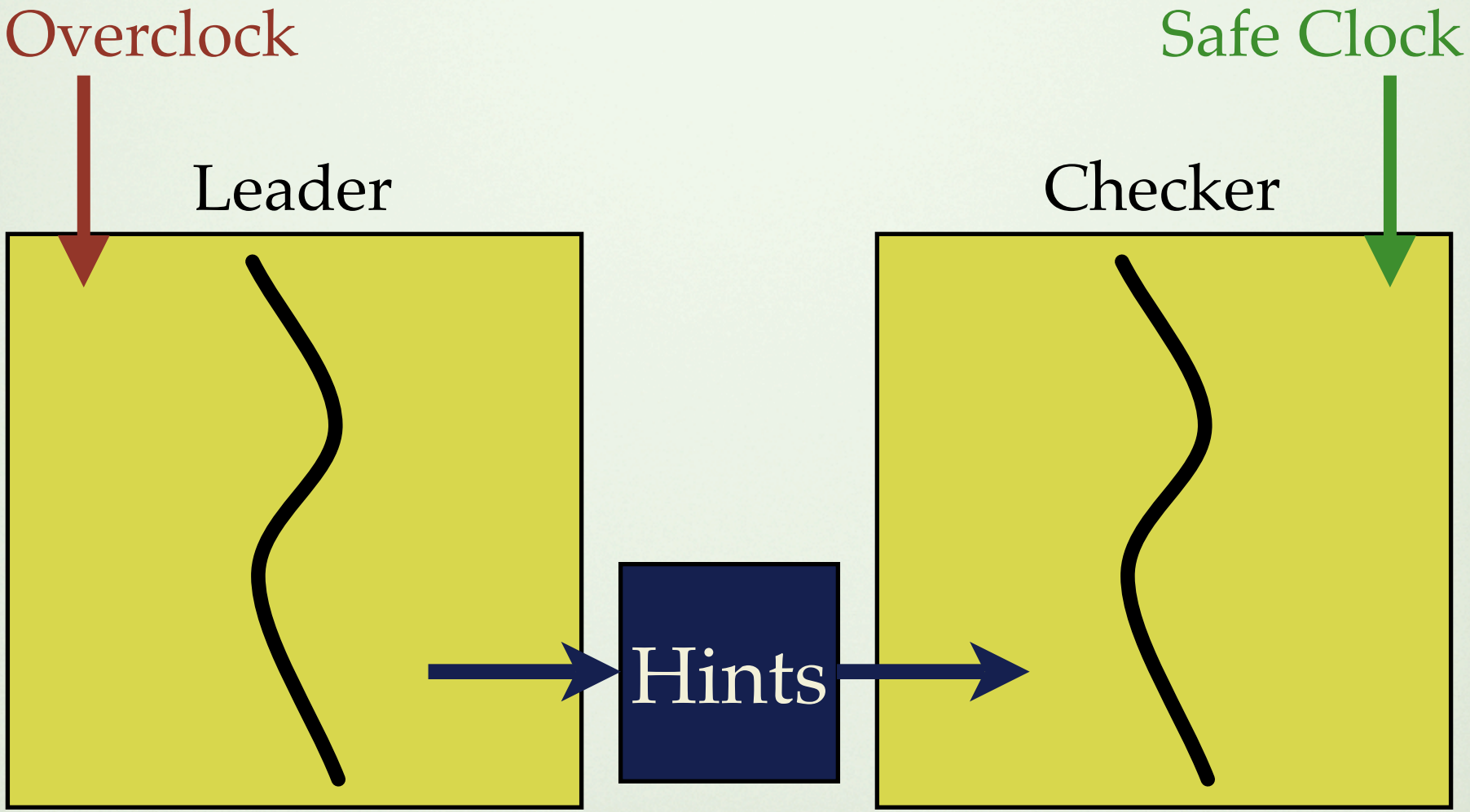
Paceline Concept



Paceline: Brian Greskamp

i-acoma group

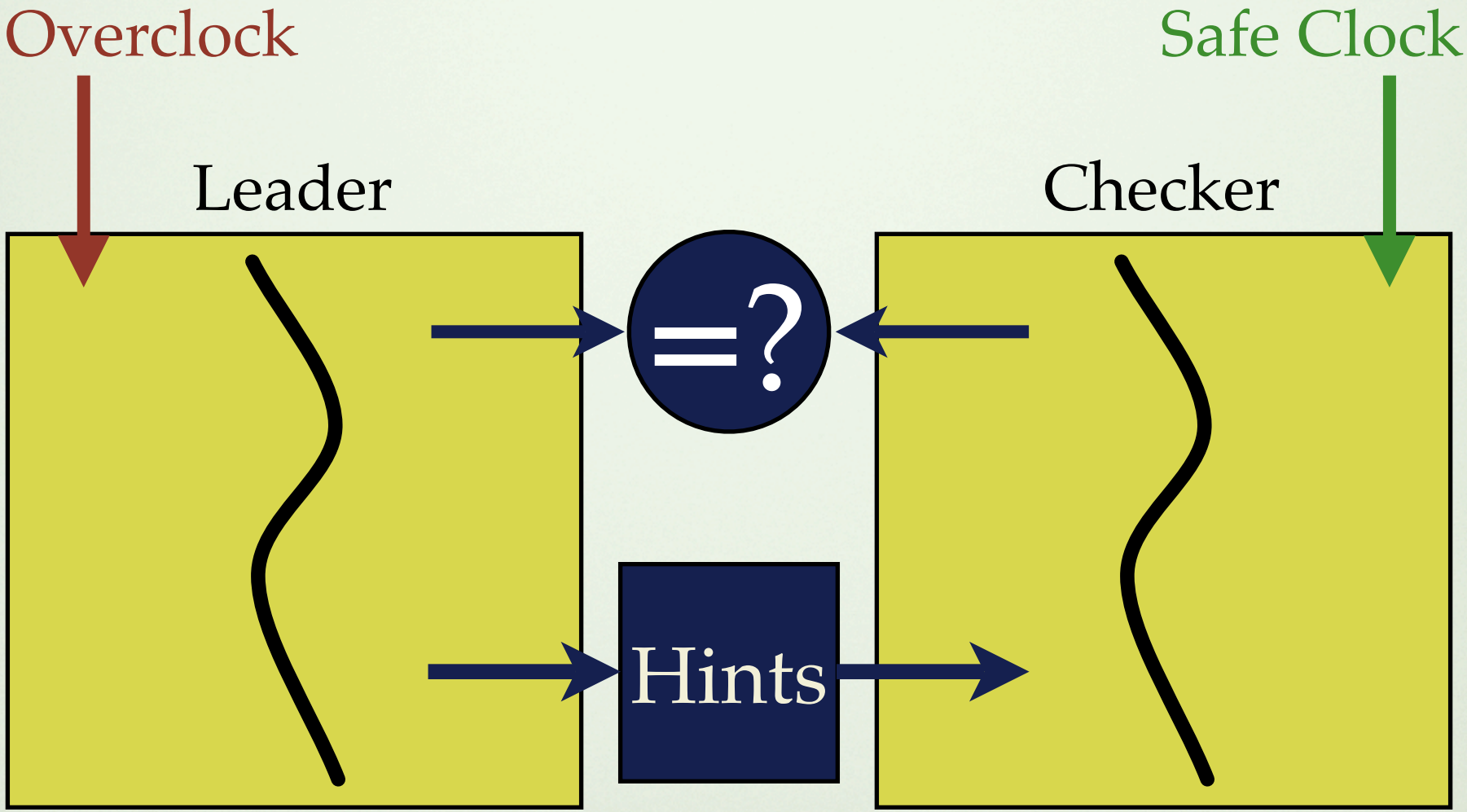
Paceline Concept



Paceline: Brian Greskamp

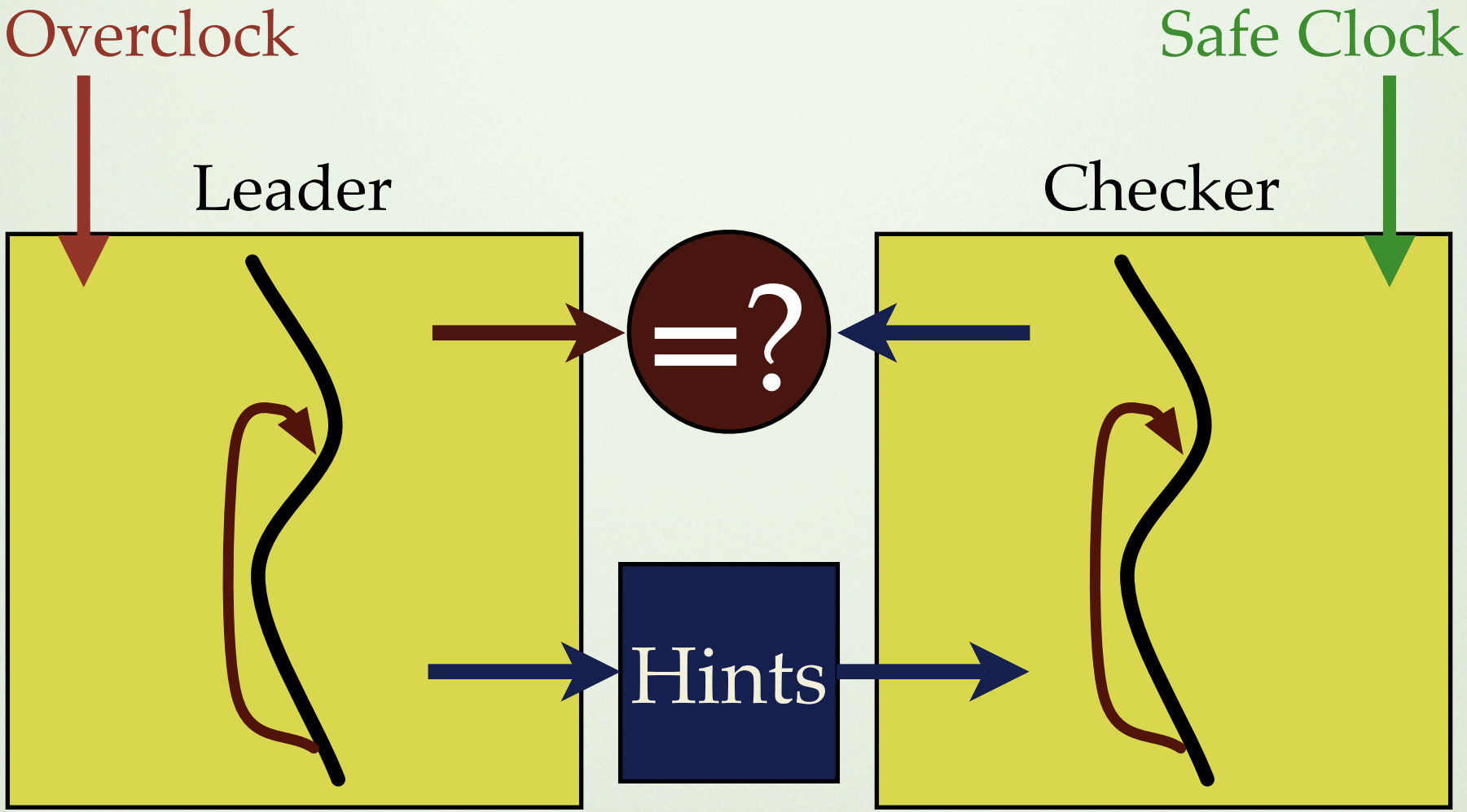
i-acoma group

Paceline Concept



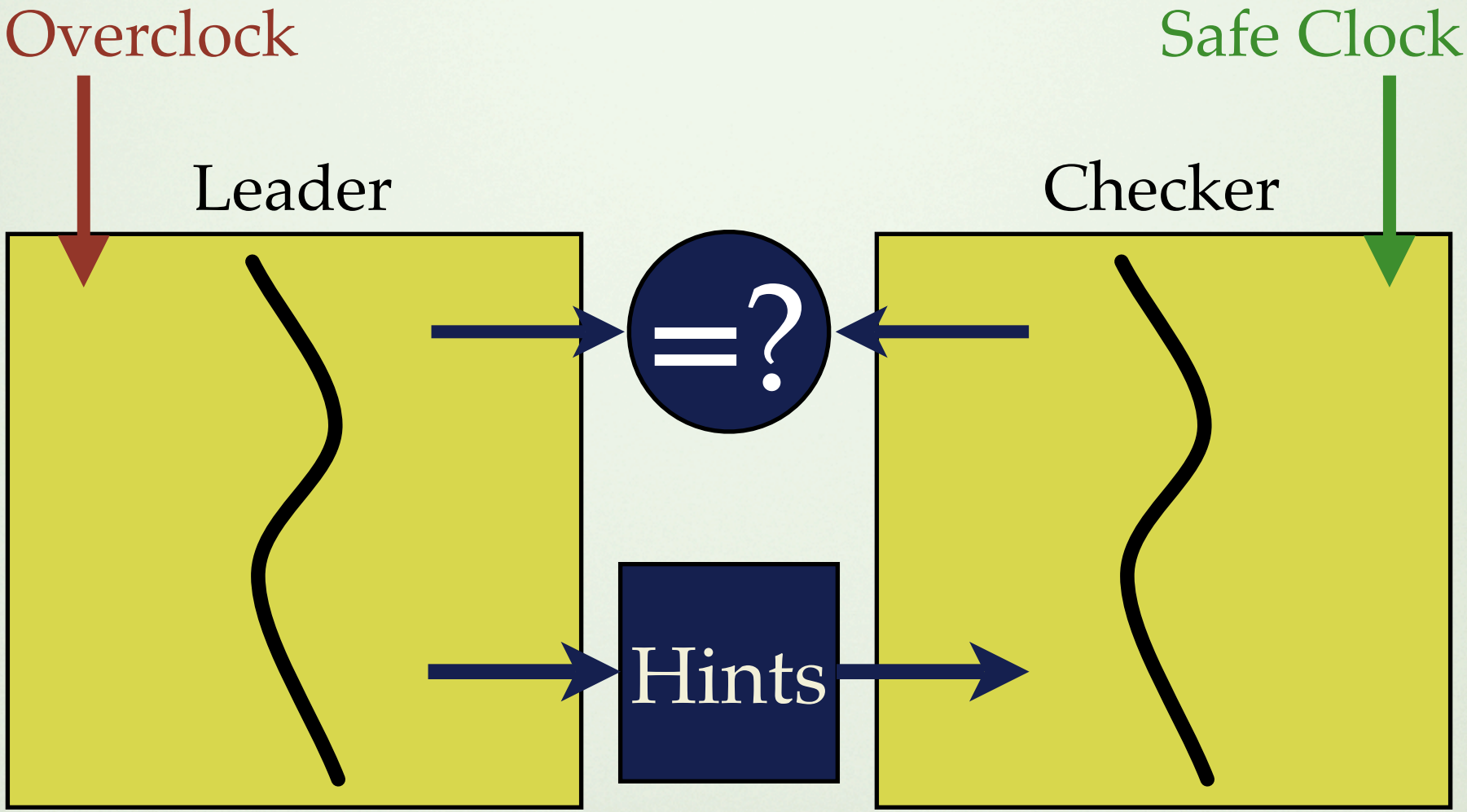
Paceline: Brian Greskamp

Paceline Concept



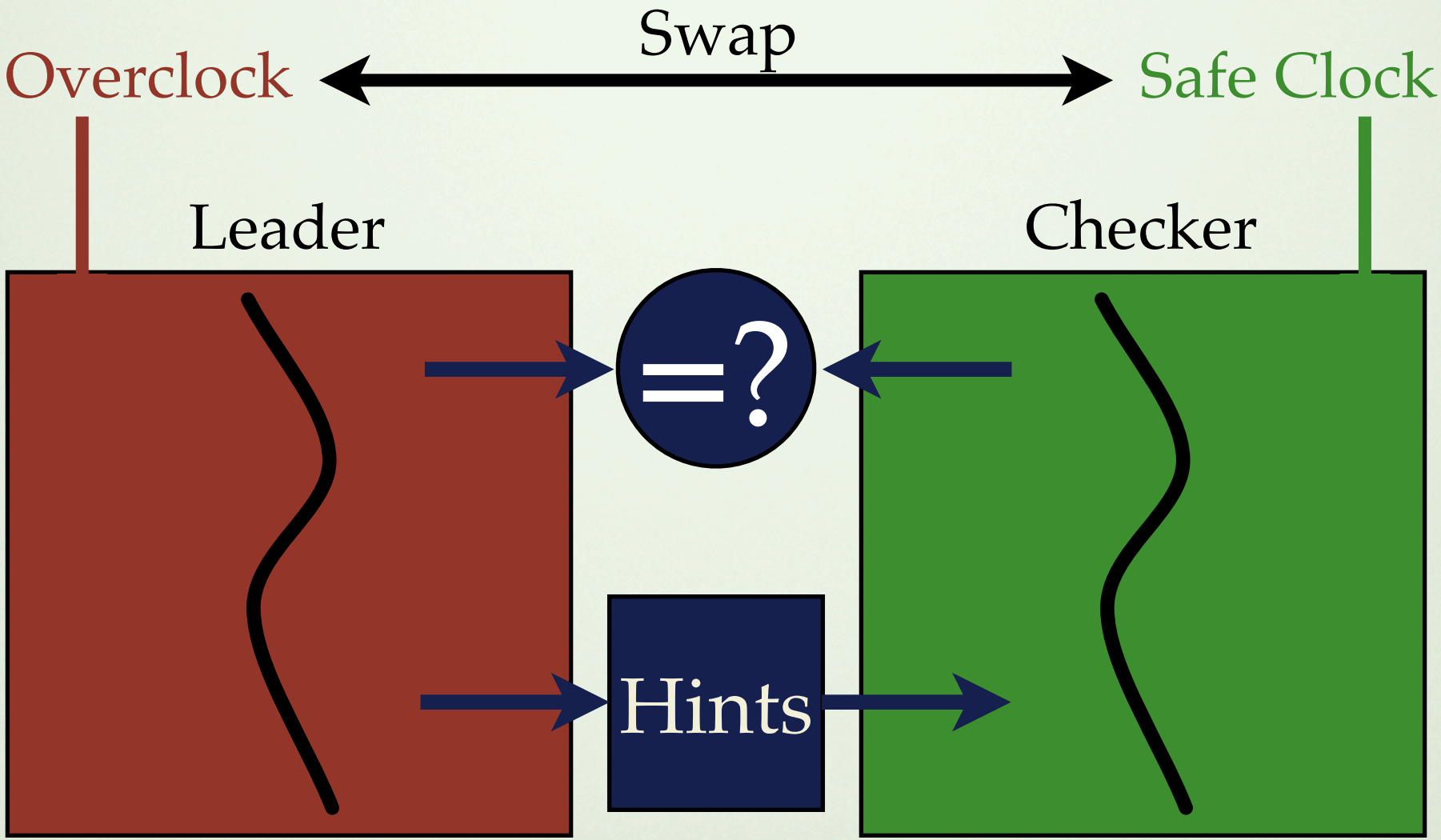
Paceline: Brian Greskamp

Paceline Concept



Paceline: Brian Greskamp

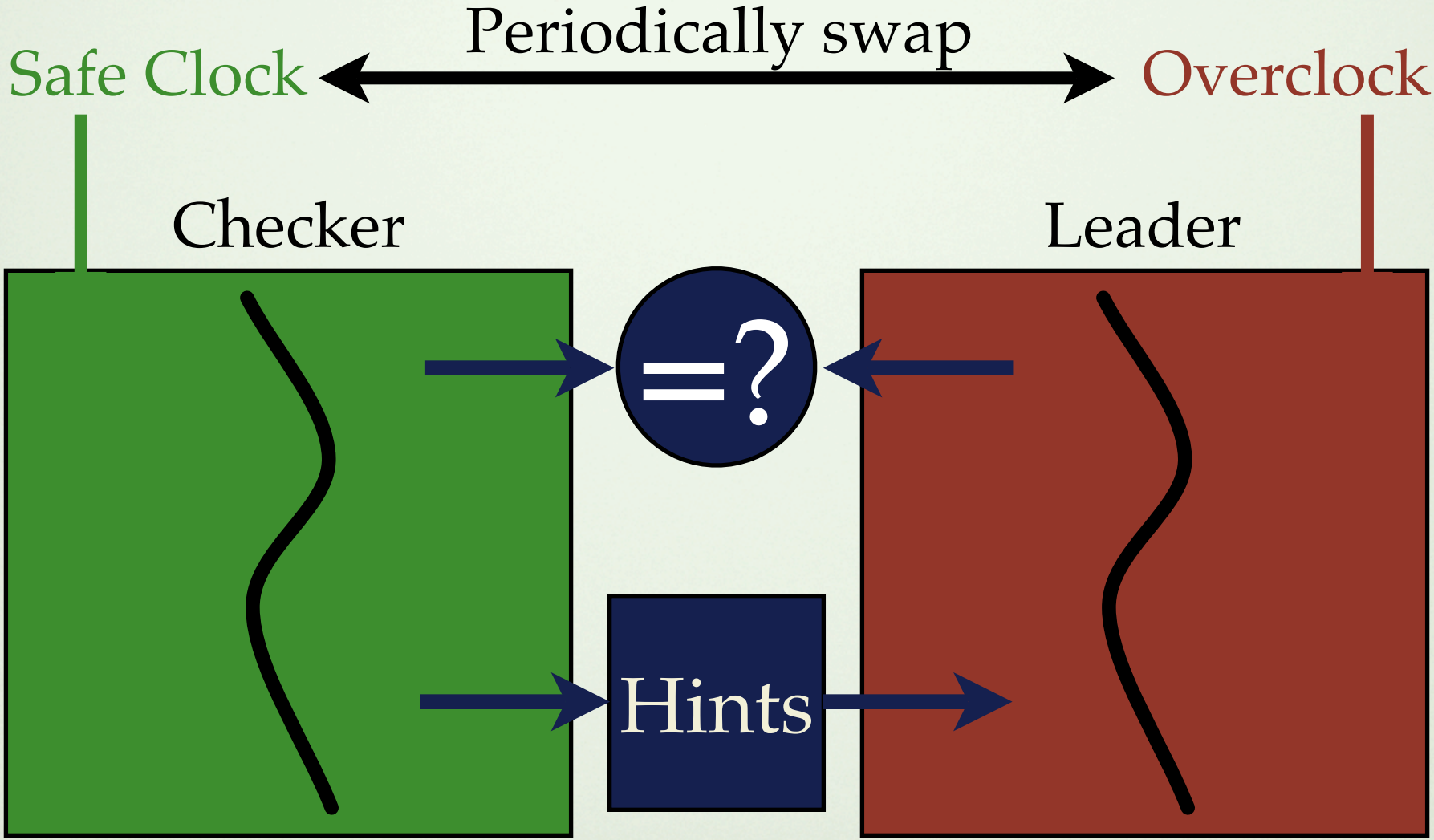
Paceline Concept



Paceline: Brian Greskamp

i-acoma group

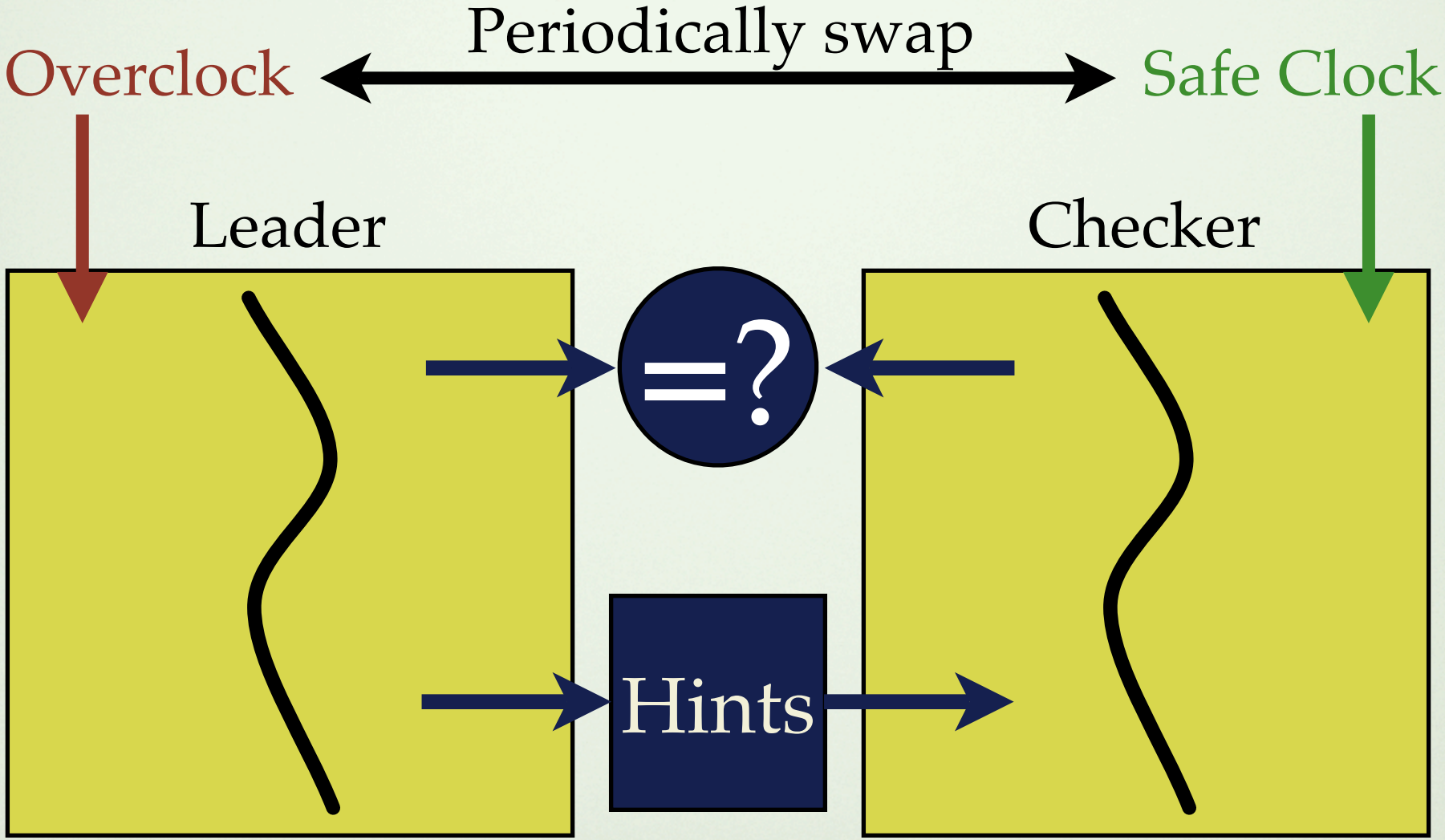
Paceline Concept



Paceline: Brian Greskamp

i-acoma group

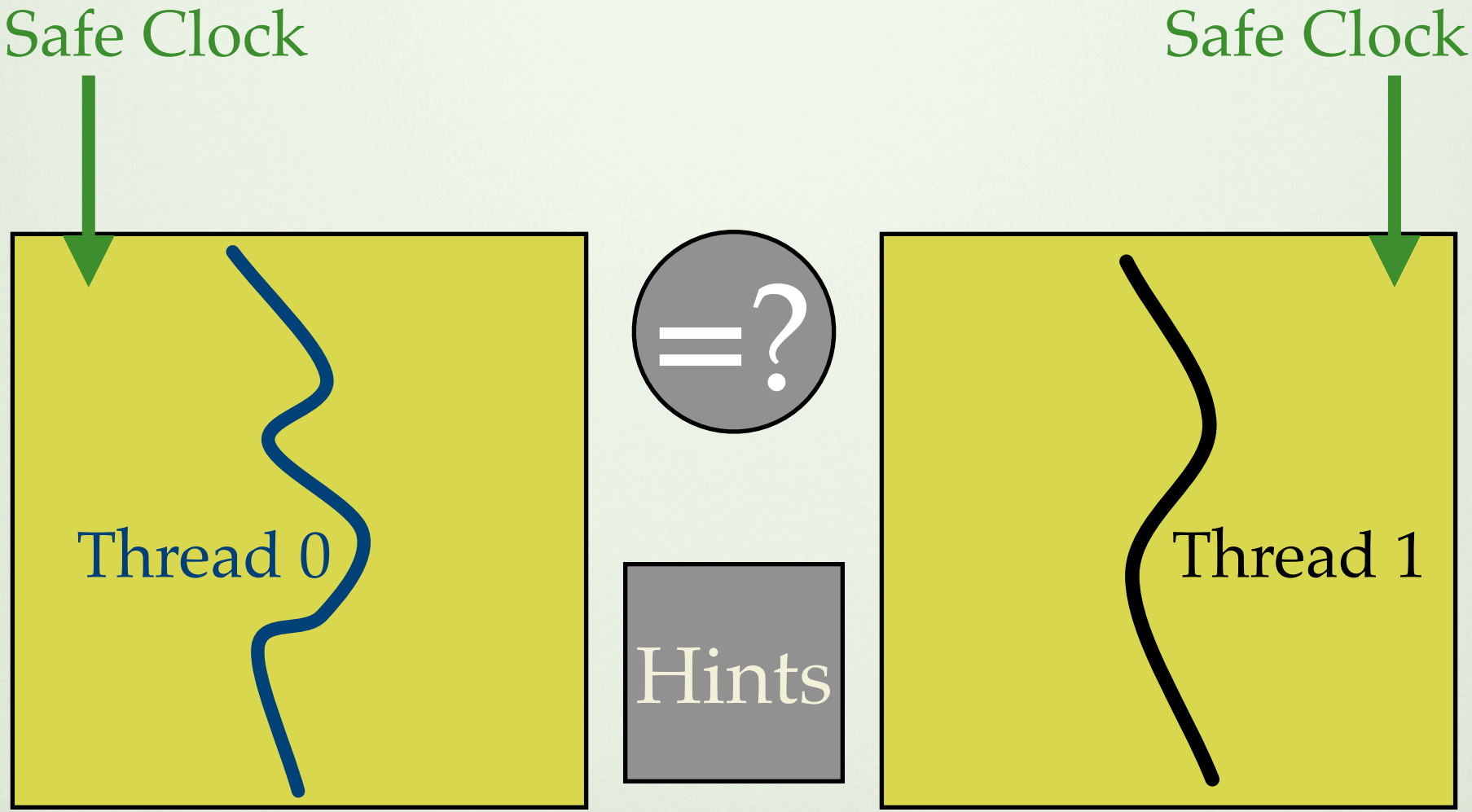
Paceline Concept



Paceline: Brian Greskamp

i-acoma group

Paceline Concept



Paceline: Brian Greskamp

i-acoma group

Contributions

- Paceline: A new approach to improving single-thread performance through replication
- Two detailed microarchitecture implementations
  - High performance
  - Minimal power and thermal impact
  - Optional additional fault tolerance

Potential for Overclocking

- Processor speed grades are determined by:
  - The slowest core on the CMP
  - ... under worst-case environmental conditions
  - ... at the point of first failure

Potential for Overclocking

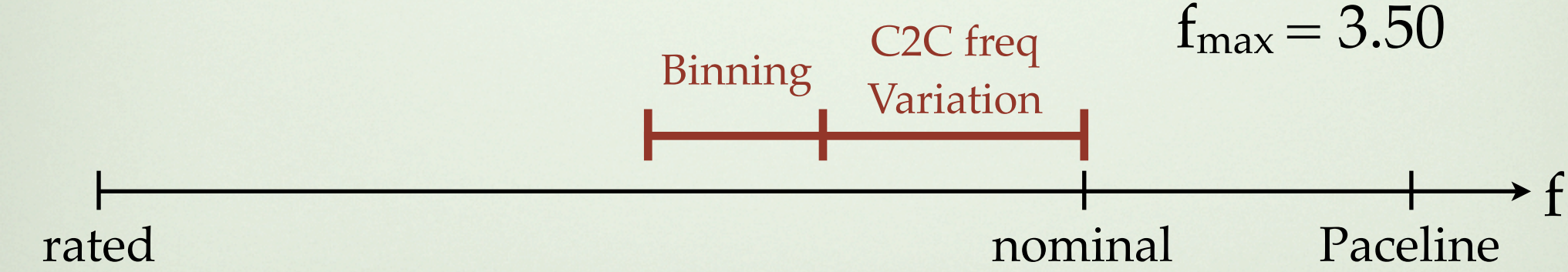
- **... the slowest core**
- Process variation causes core-to-core frequency variation
- Speed grading places all chips into pre-determined bins

CMP with Variation

3.62	4.22	4.05
3.71	4.15	3.81
4.05	3.59	3.90

~~$f_{max} = 3.62$~~

$f_{max} = 3.50$

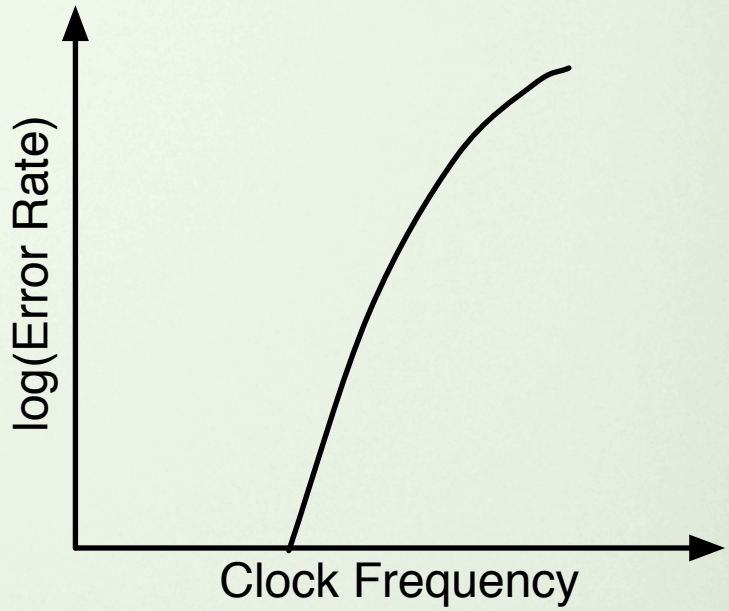


Potential for Overclocking

- **... under worst case conditions**
- Circuits slow with:
  - Age (NBTI, HCI)
  - Increasing temperature (activity, environment)
  - Supply voltage droop (*L dI/dt*, *IR*)

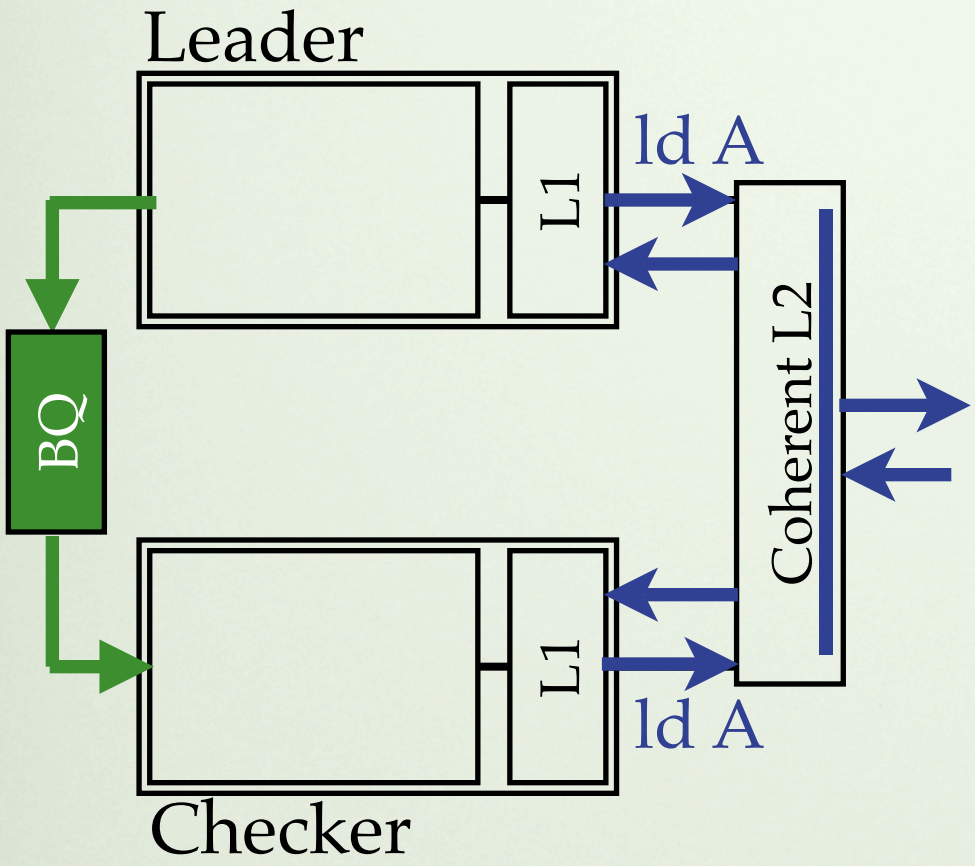
Overclockability

- **... at the point of first failure**
- Redundancy can detect and correct occasional timing faults



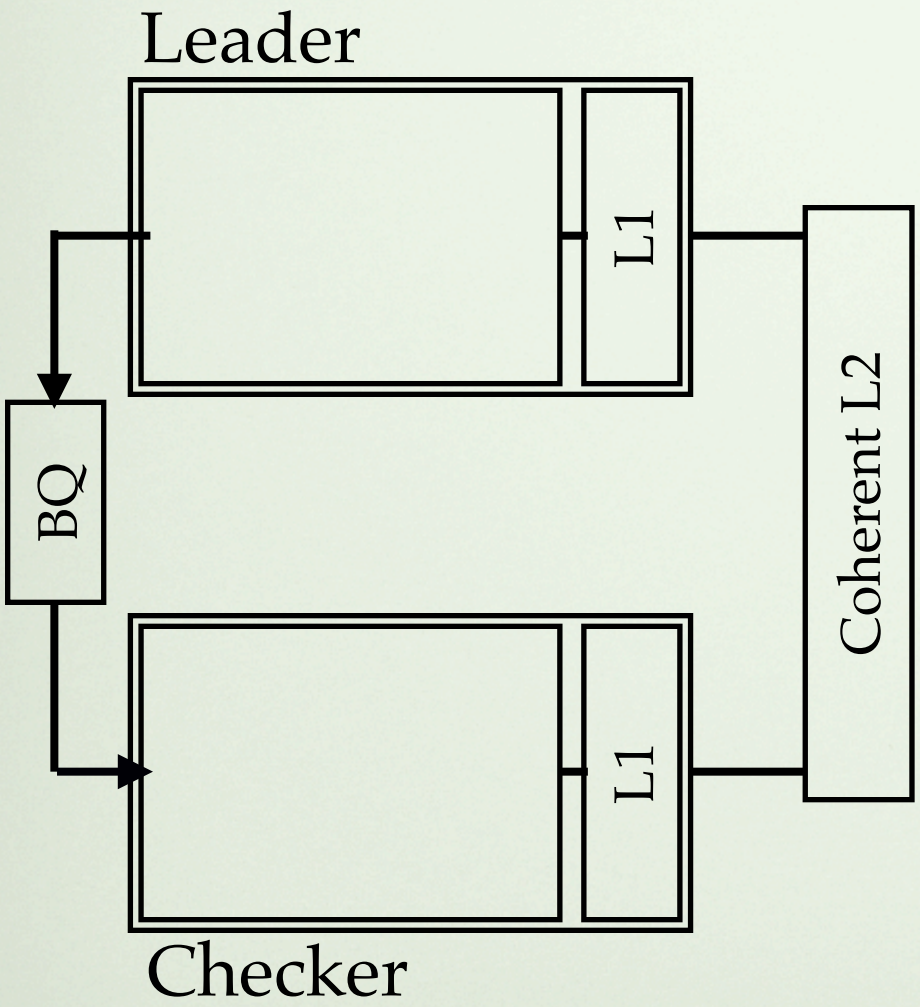
i-acoma group

Paceline: Achieving Speedup

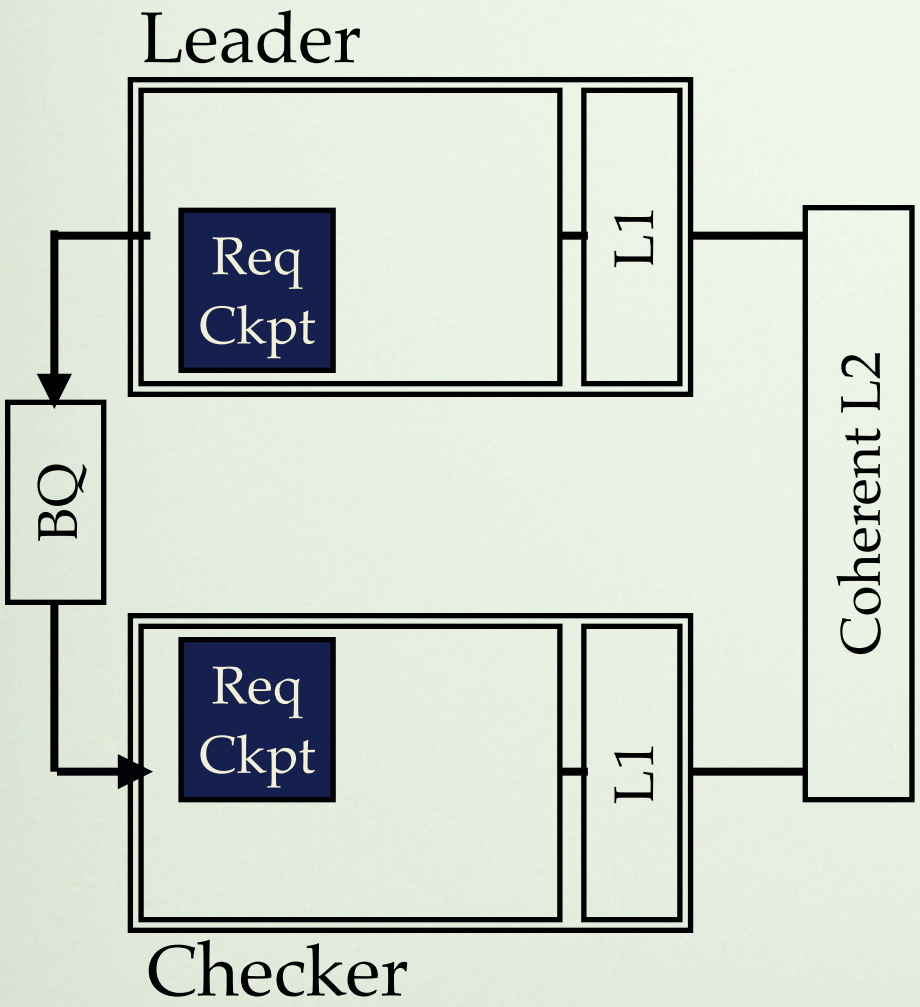


- Note: Cores don't operate in lockstep; leader runs ahead
- Hints from leader improve checker performance (IPC)
  - Branch outcomes
  - Prefetches

Paceline: Ensuring Correctness

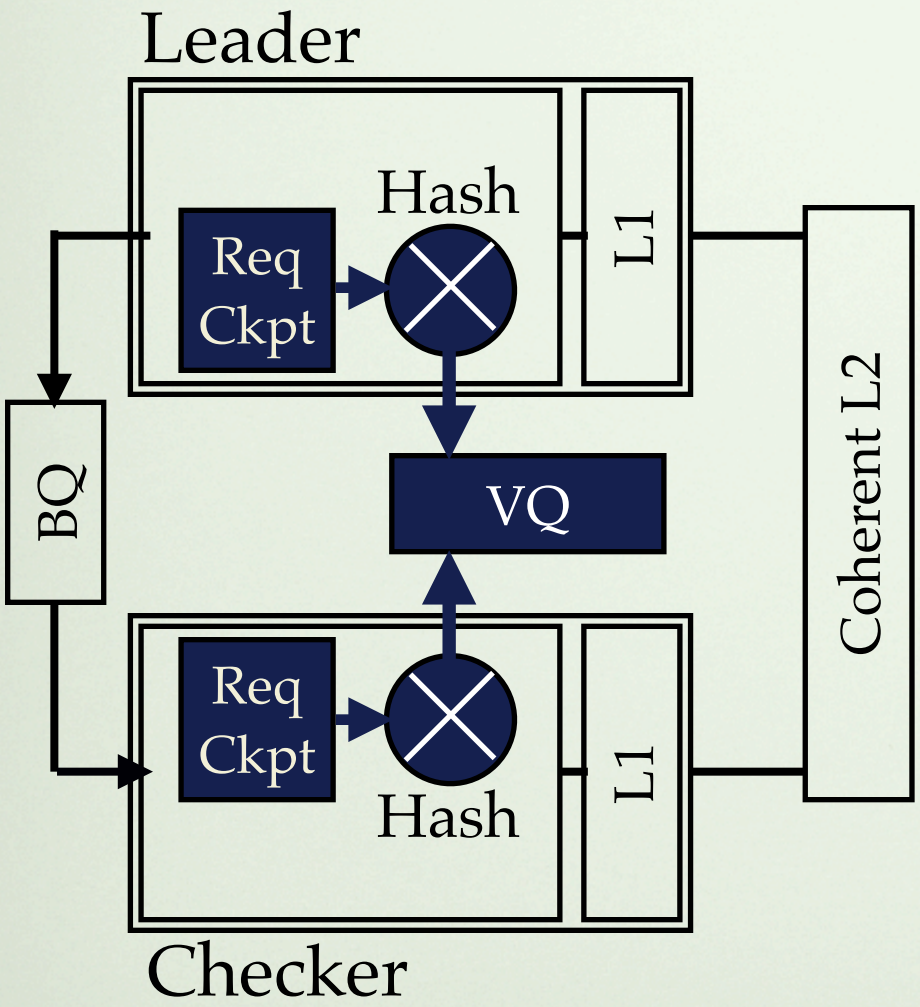


Paceline: Ensuring Correctness



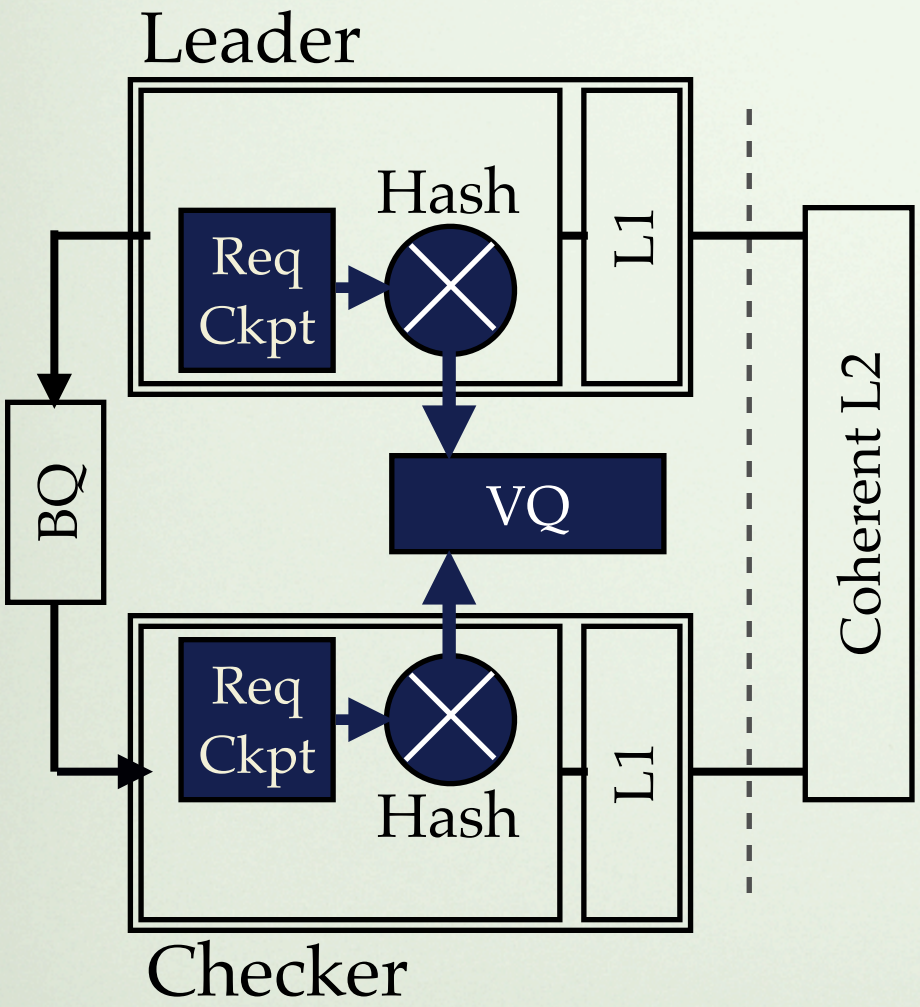
- Periodically take register checkpoints

Paceline: Ensuring Correctness



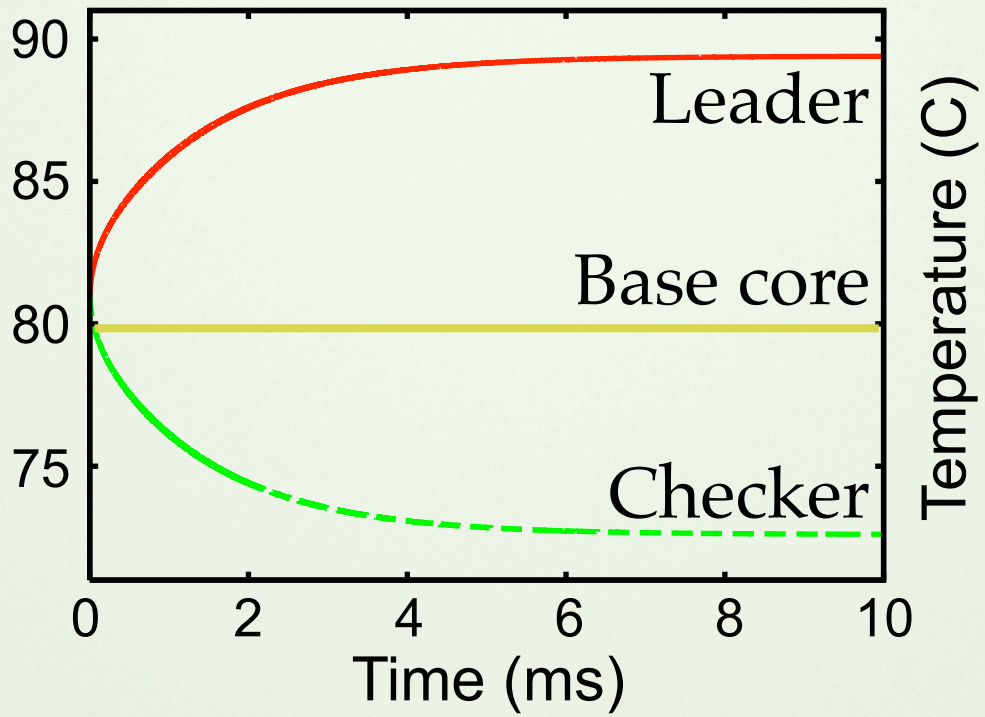
- Periodically take register checkpoints
- Add VQ to compare leader and checker states

Paceline: Ensuring Correctness



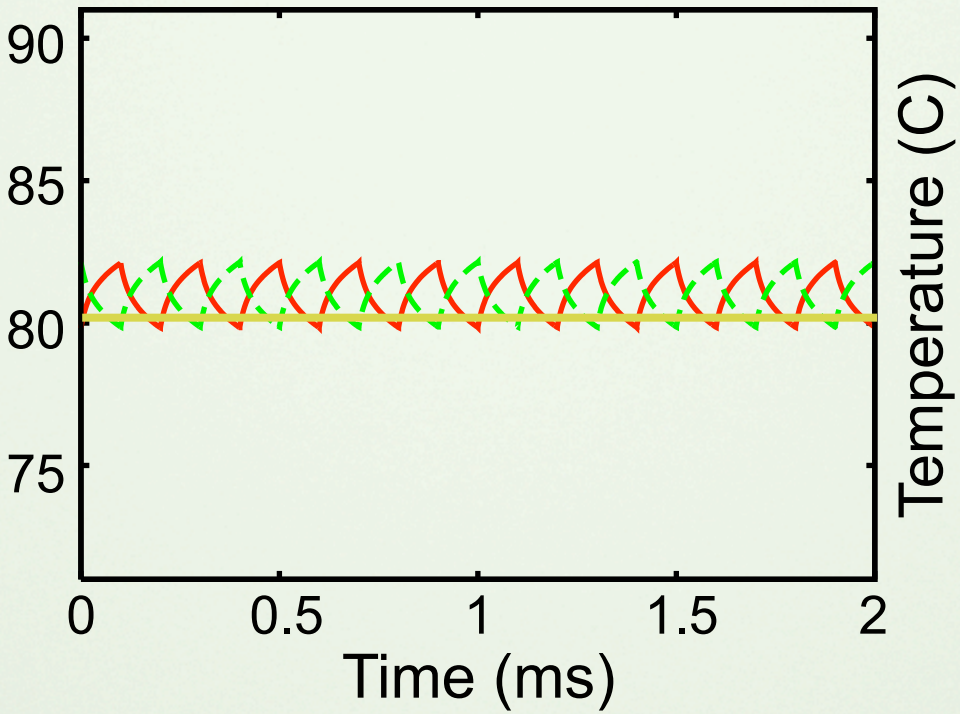
- Periodically take register checkpoints
- Add VQ to compare leader and checker states
- No I/O or write is performed at L2 until it has been checked

Paceline: Thermal Management



- Leader consumes more power than a base core
- Often, checker consumes less power than base core

Paceline: Thermal Management

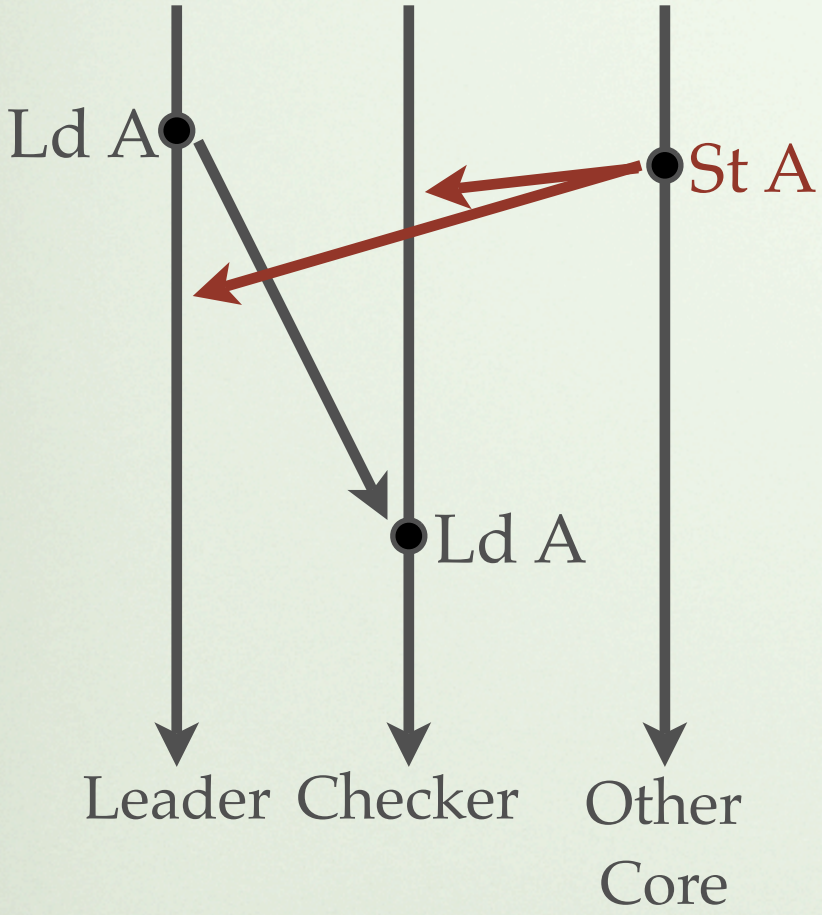


- Periodically swapping cores averages temperature
  - Improves reliability, overclocking potential
  - Minimizes leakage power

Types of Faults

Fault Type	Repeats during Re-Executions?	Leader Impact	Checker Impact
Timing Error due to Overclock	Likely	Register, L1 State	None

Input Incoherence



- Both cores' L1s send normal read misses to L2
- Other core's write can cause leader and checker to read different values ("incoherence")[Slipstream02][Reunion06]

Types of Faults

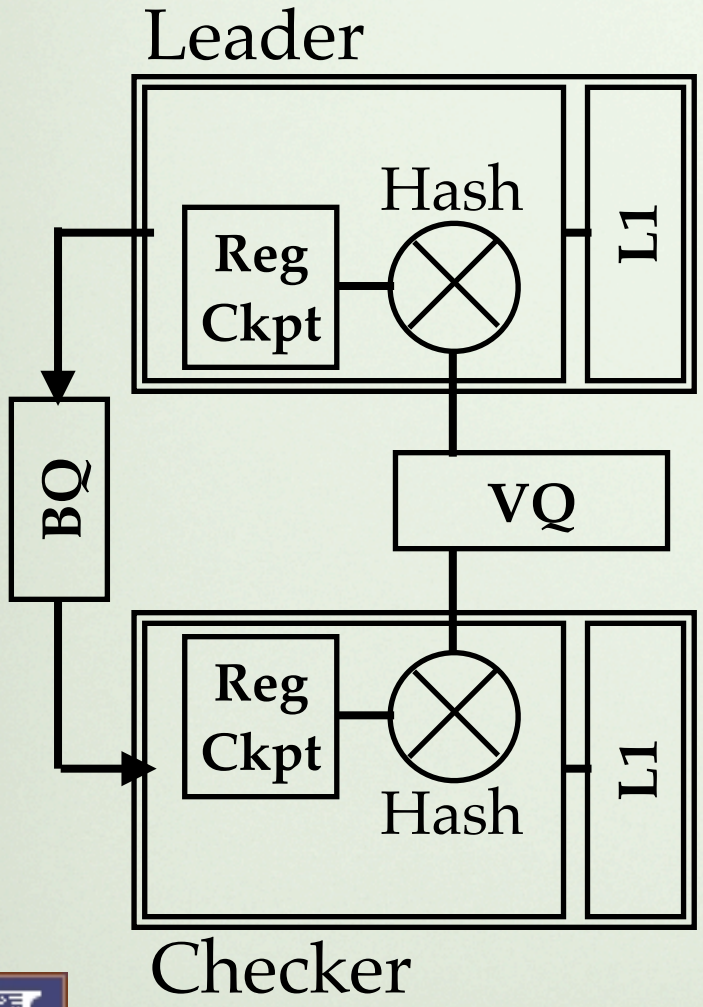
Fault Type	Repeats during Re-Executions?	Leader Impact	Checker Impact
Timing Error due to Overclock	Likely	Register, L1 State	None
Input Incoherence	Possibly	Register, L1 State	None
Soft Error	No	Register, L1 State	Register, L1 State

Paceline Architecture

- Two Paceline design points:
  - *Simple* guarantees correct execution if checker has no soft errors
  - *High-reliability* tolerates checker soft errors
- Both provide improved performance
- Both tolerate all leader faults

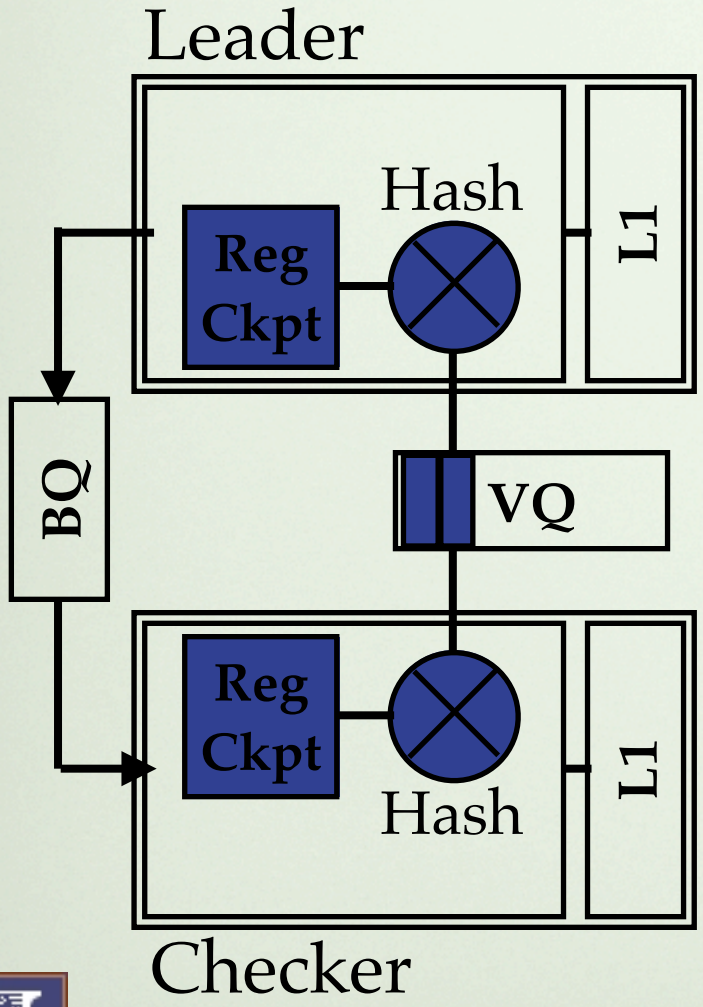
Simple Microarchitecture

Assume Checker is always correct



Simple Microarchitecture

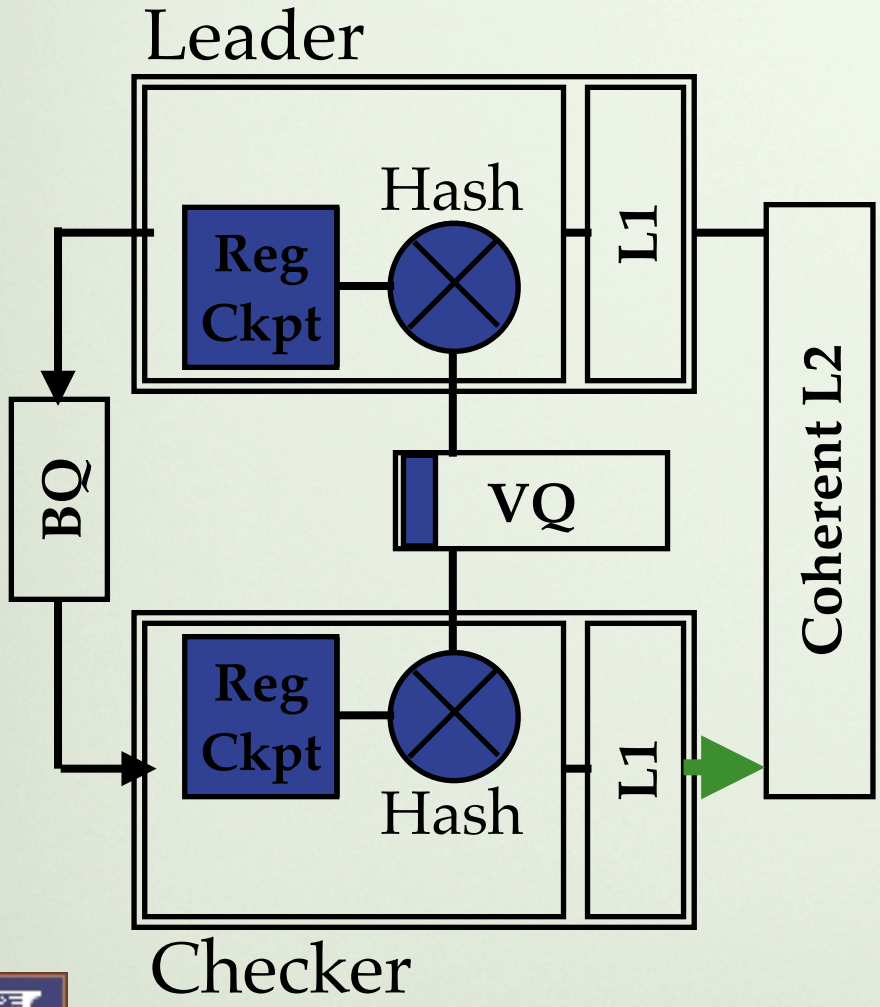
Assume Checker is always correct



- VQ buffers and compares register value hashes

Simple Microarchitecture

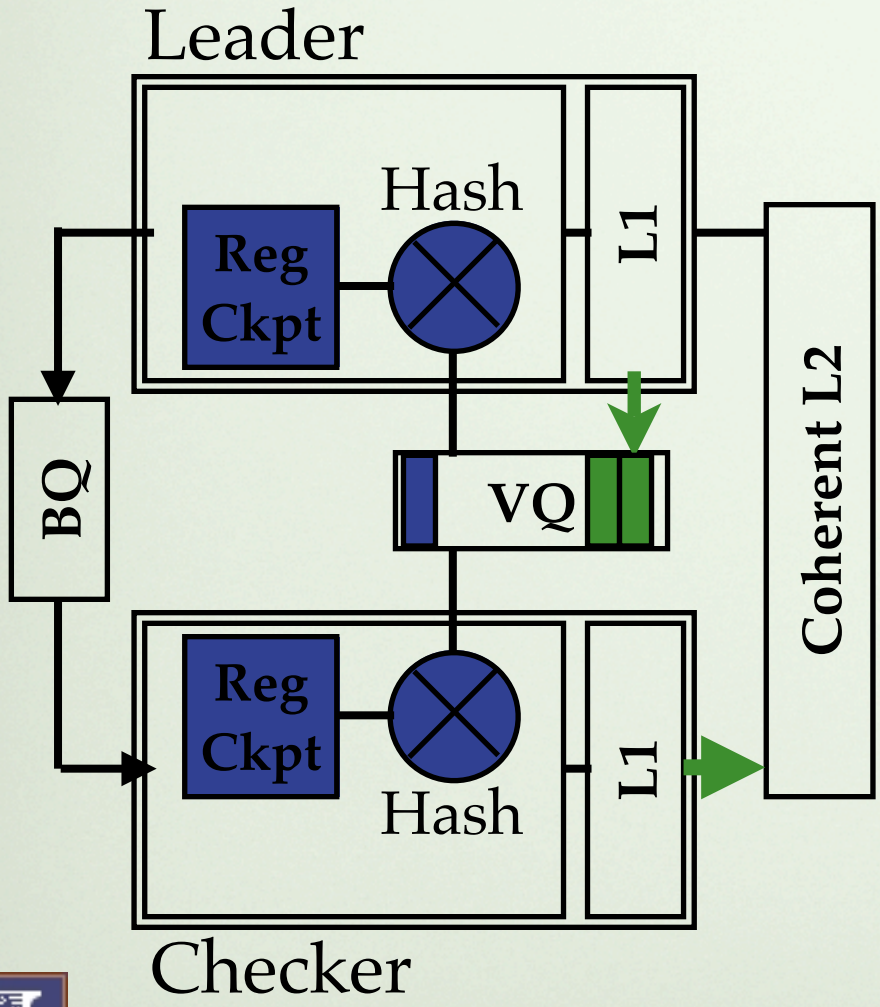
Assume Checker is always correct



- VQ buffers and compares register value hashes
- Only checker writes to L2

Simple Microarchitecture

Assume Checker is always correct

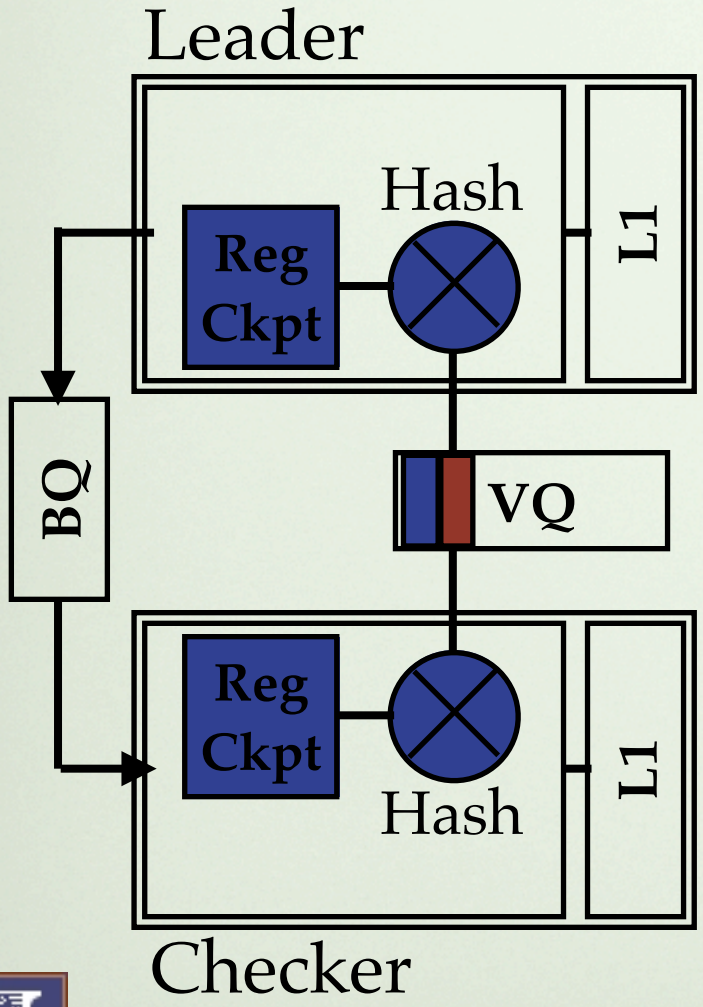


- VQ buffers and compares register value hashes
- Only checker writes to L2
- Leader write handling options
  - Drop writebacks
  - Use VQ as victim cache

Simple Microarchitecture: Recovery

Assume Checker is always correct

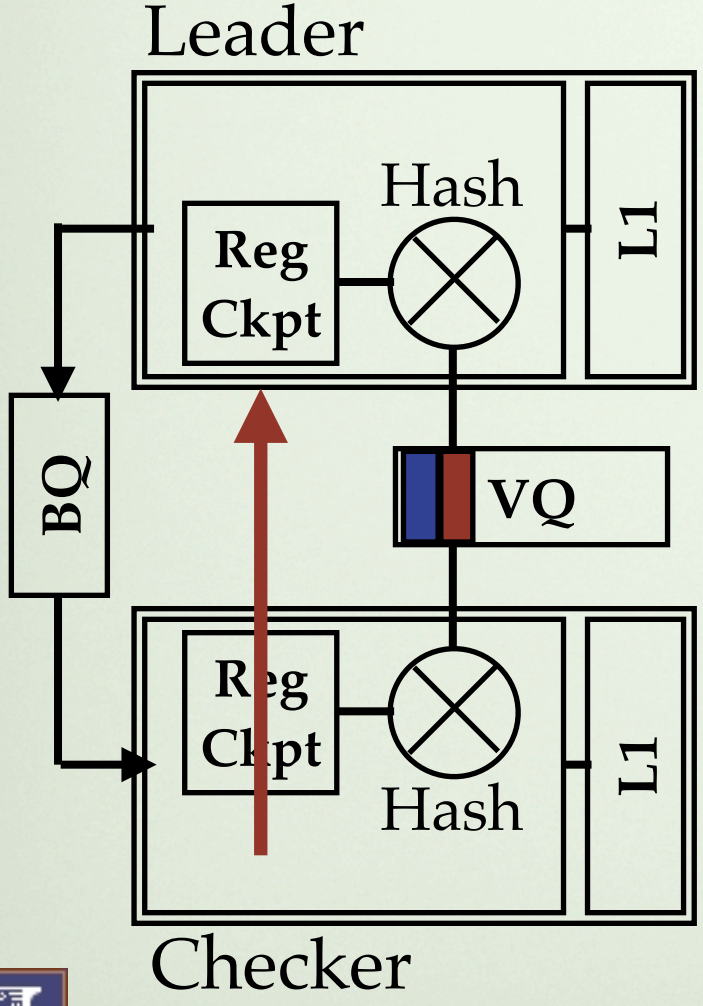
→ **On error, copy state from Checker**



Simple Microarchitecture: Recovery

Assume Checker is always correct

→ **On error, copy state from Checker**

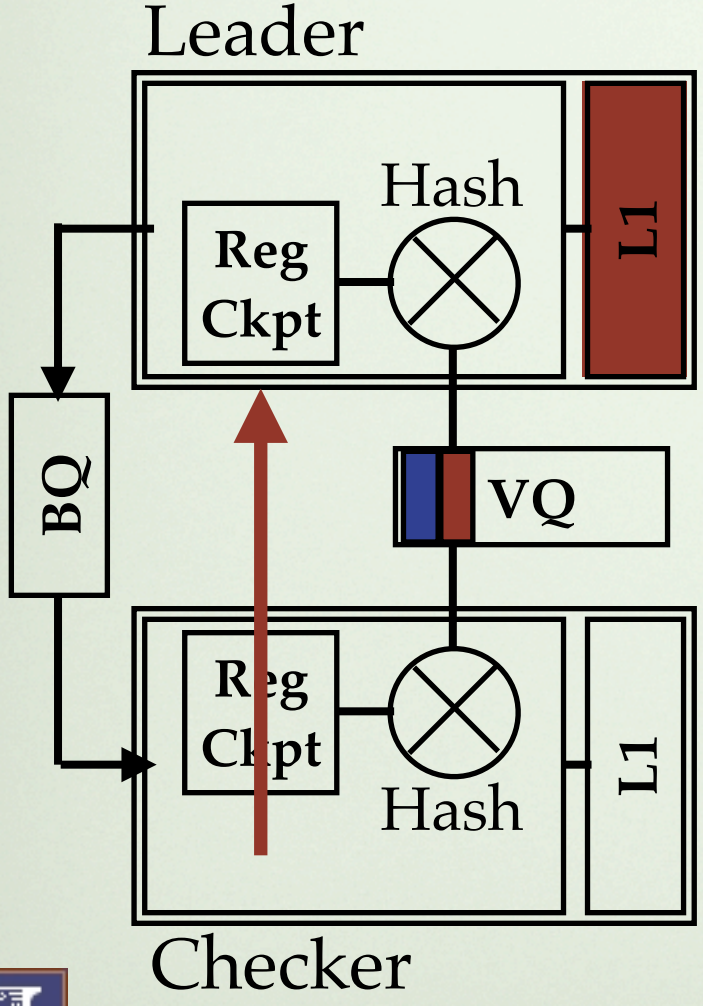


- Roll leader *forward* past fault
  1. Copy checker regfile contents to leader

Simple Microarchitecture: Recovery

Assume Checker is always correct

→ **On error, copy state from Checker**

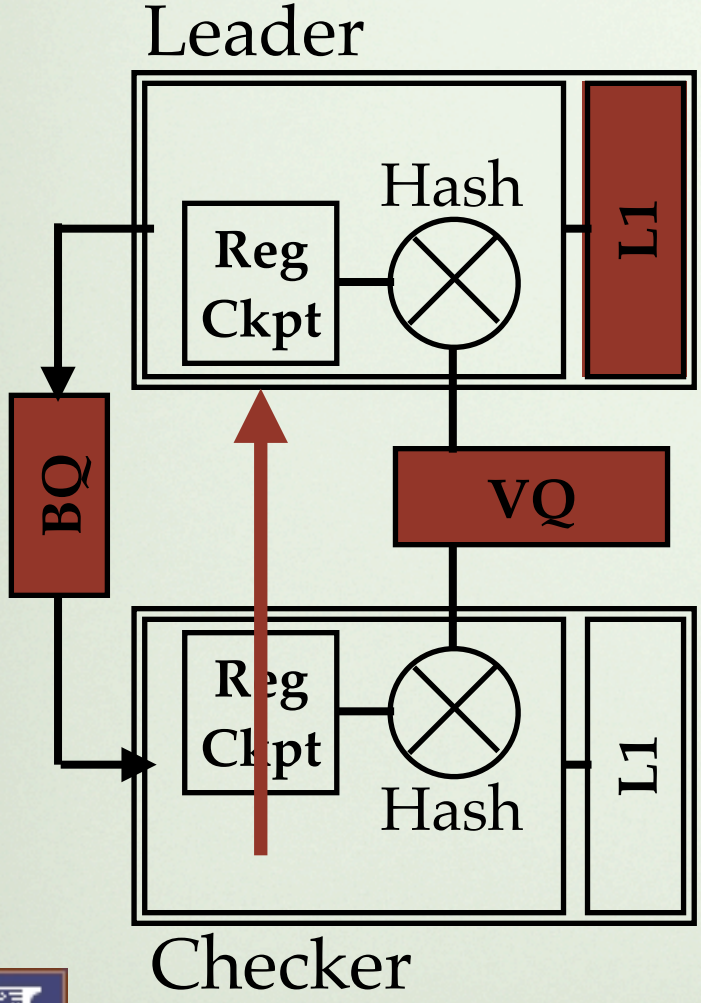


- Roll leader *forward* past fault
  1. Copy checker regfile contents to leader
  2. Flush leader L1 cache

Simple Microarchitecture: Recovery

Assume Checker is always correct

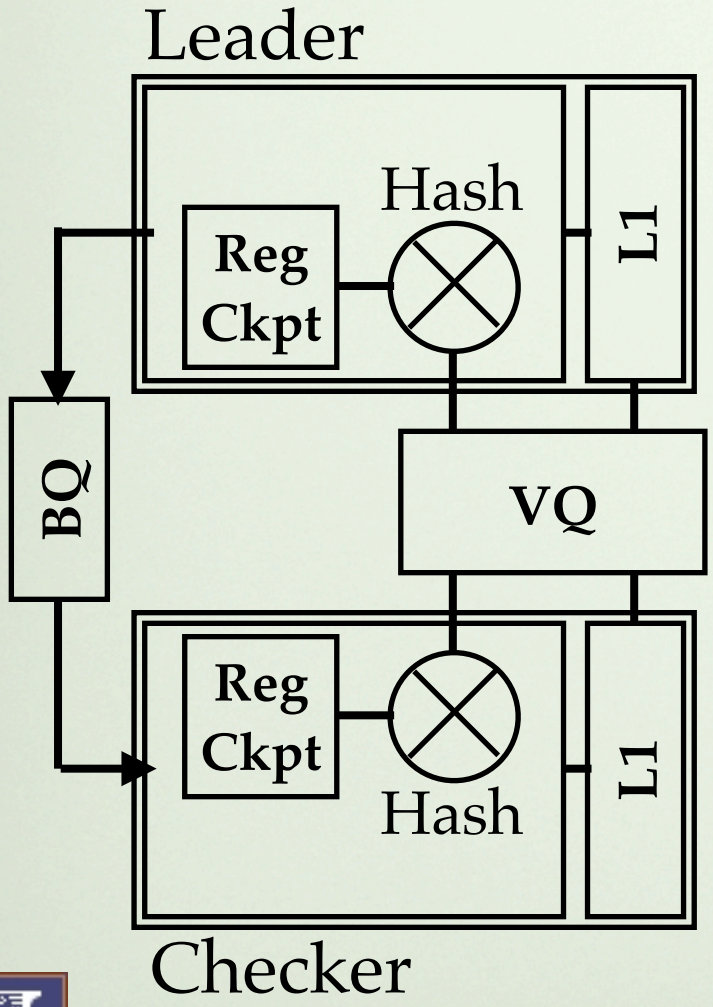
→ **On error, copy state from Checker**



- Roll leader *forward* past fault
  1. Copy checker regfile contents to leader
  2. Flush leader L1 cache
  3. Clear VQ, BQ

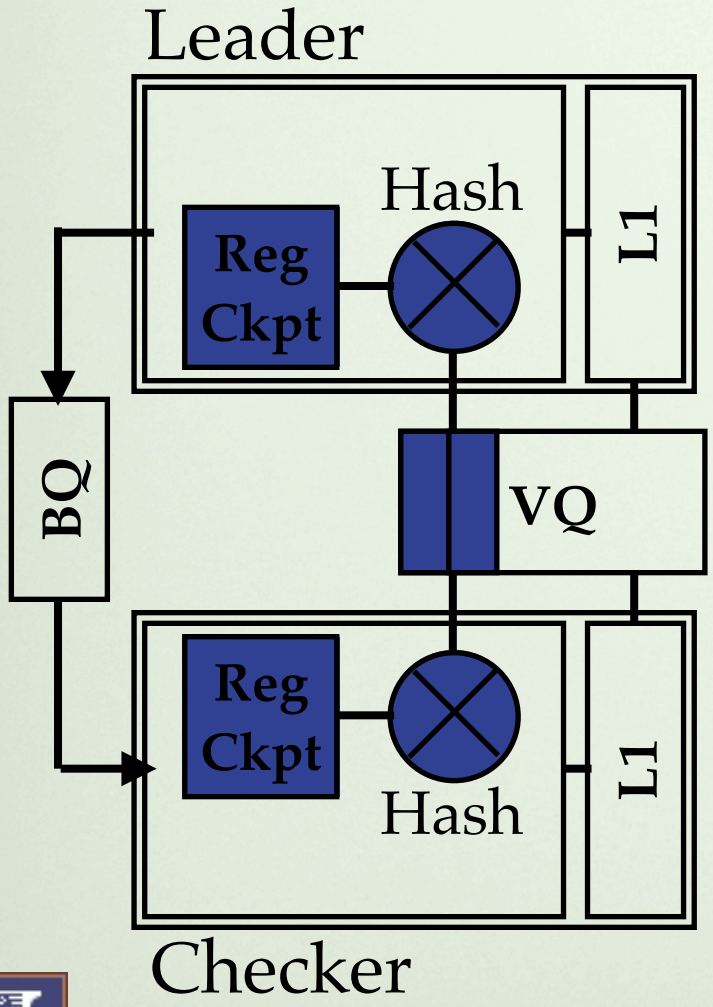
High Reliability Microarchitecture

Assume Checker can experience soft errors



High Reliability Microarchitecture

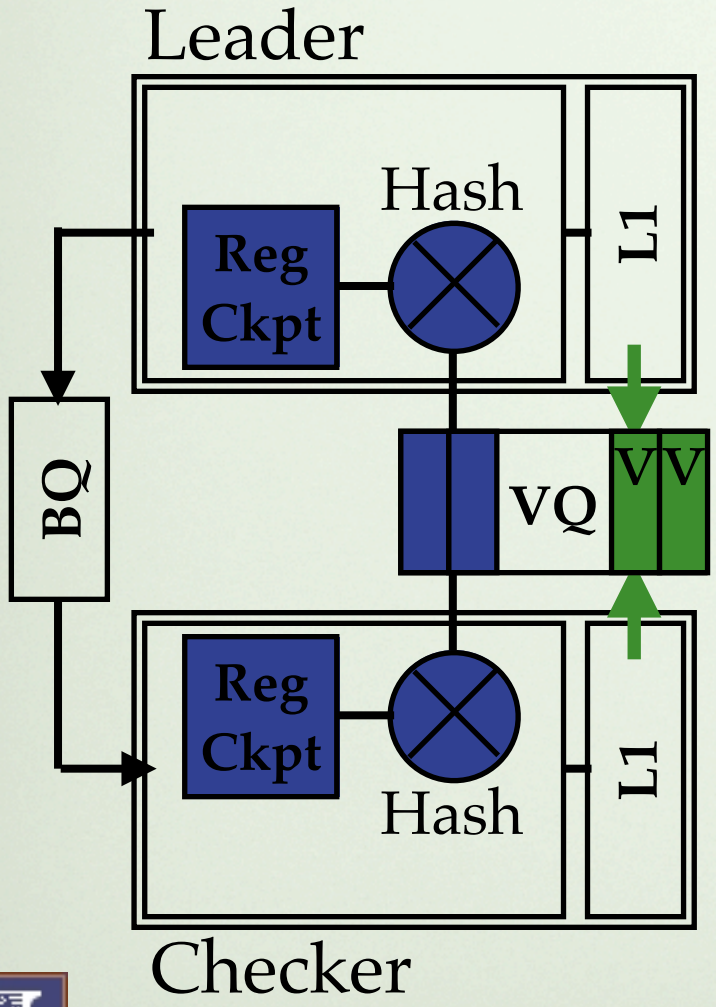
Assume Checker can experience soft errors



- VQ buffers and compares register hashes

High Reliability Microarchitecture

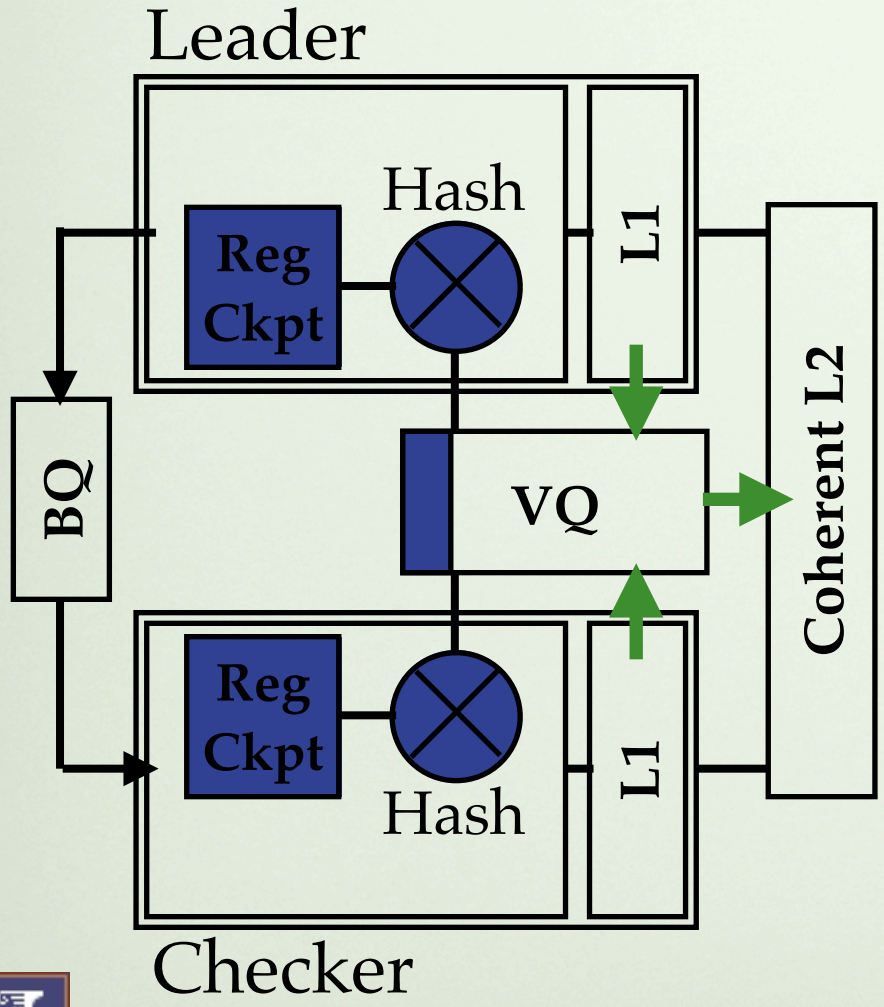
Assume Checker can experience soft errors



- VQ buffers and compares register hashes
- VQ buffers and compares write values
  - L1s are Write-Through

High Reliability Microarchitecture

Assume Checker can experience soft errors

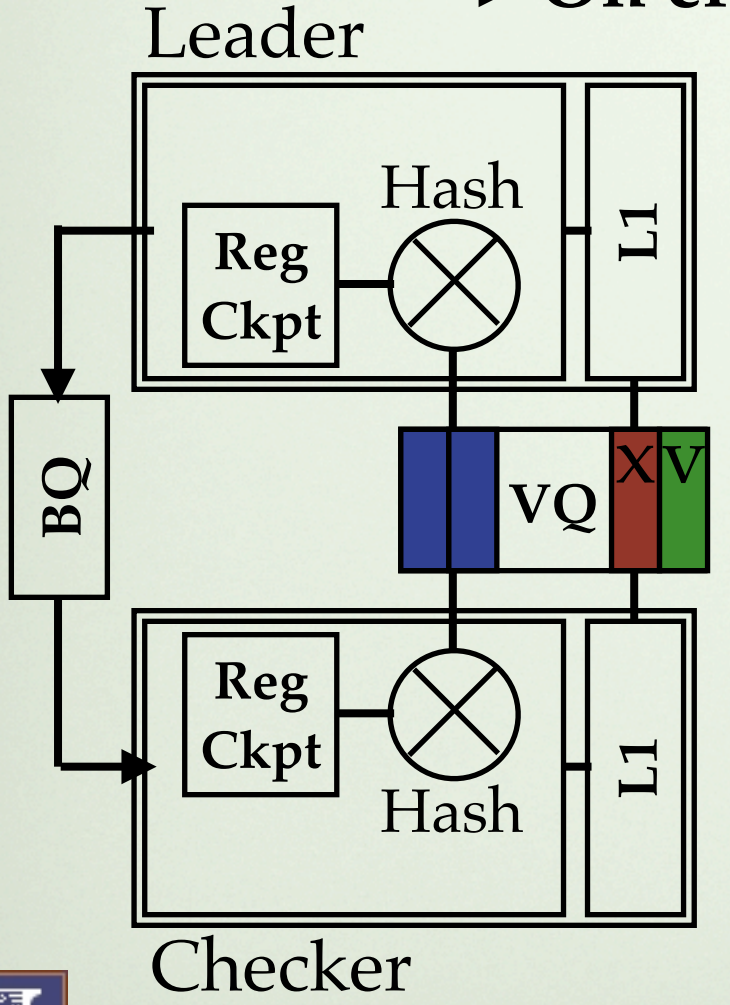


- VQ buffers and compares register hashes
- VQ buffers and compares write values
  - L1s are Write-Through
- VQ releases writes to L2 after successful comparison of checkpoint interval

High Reliability Microarchitecture: Recovery

Assume Checker can experience soft errors

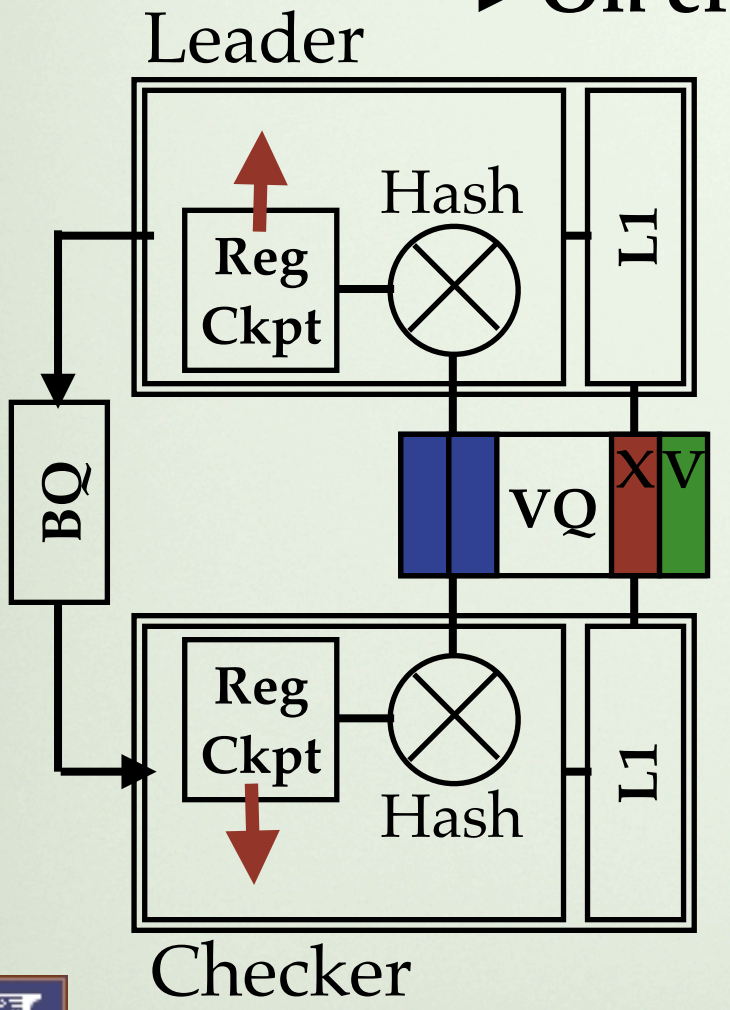
→ **On error, roll back both cores**



High Reliability Microarchitecture: Recovery

Assume Checker can experience soft errors

→ **On error, roll back both cores**

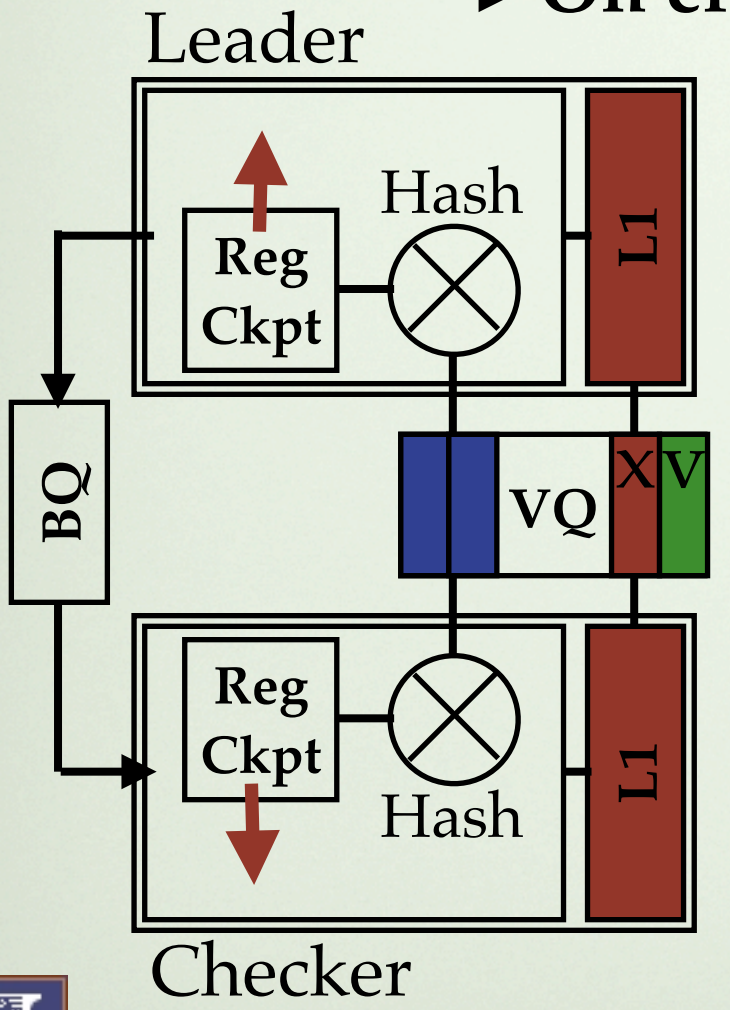


1. Restore register checkpoint in both cores

High Reliability Microarchitecture: Recovery

Assume Checker can experience soft errors

→ **On error, roll back both cores**

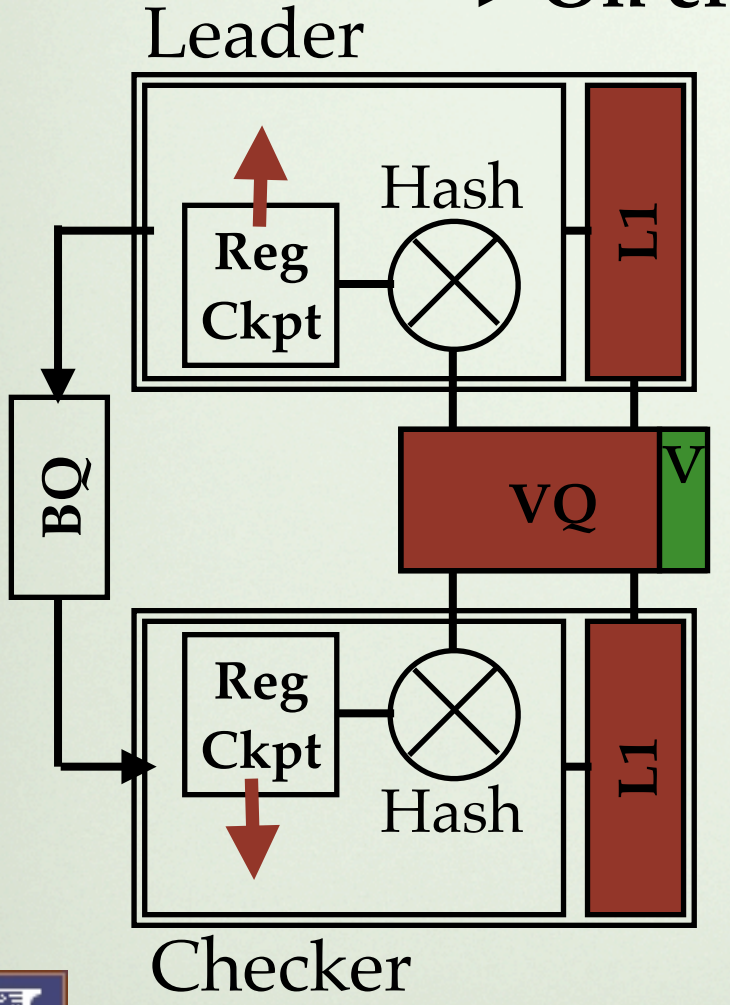


1. Restore register checkpoint in both cores
2. Flush both caches

High Reliability Microarchitecture: Recovery

Assume Checker can experience soft errors

→ **On error, roll back both cores**

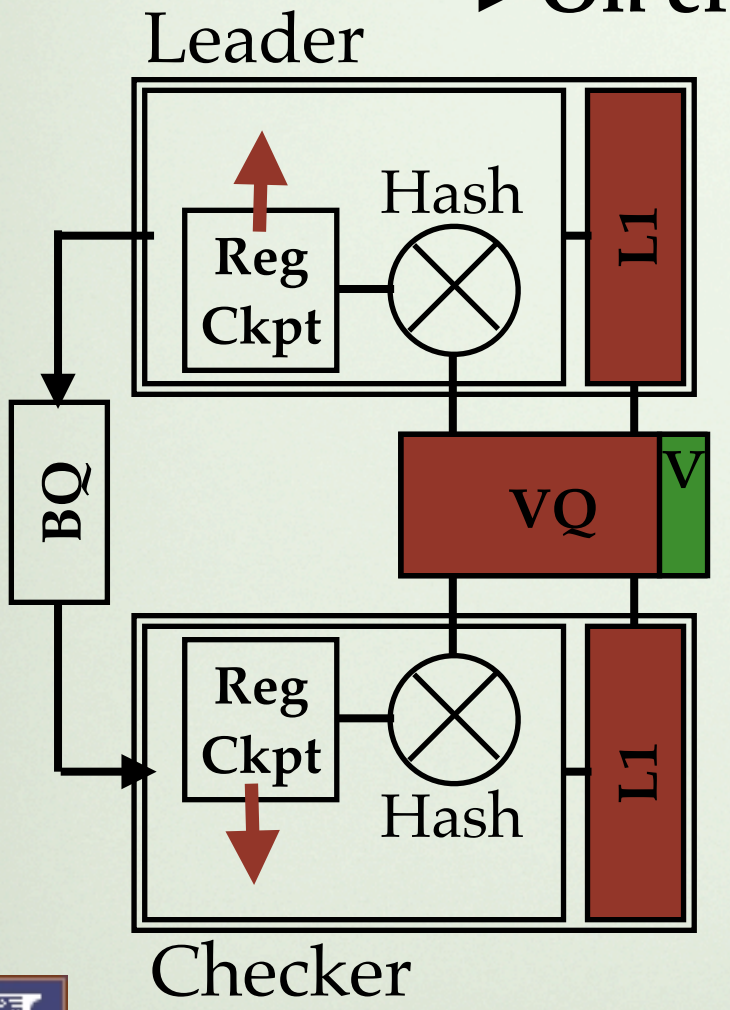


1. Restore register checkpoint in both cores
2. Flush both caches
3. Invalidate all entries in VQ from the failing checkpoint

High Reliability Microarchitecture: Recovery

Assume Checker can experience soft errors

→ **On error, roll back both cores**



1. Restore register checkpoint in both cores
2. Flush both caches
3. Invalidate all entries in VQ from the failing checkpoint
4. Restart both cores with safe clock frequency

Special Cases

- Interrupt: Deliver at next checkpoint
- Read-Modify-Write: Leader sends read to prefetch the line, but only checker can perform write
- Repeated incoherence: Techniques to avoid livelock in paper

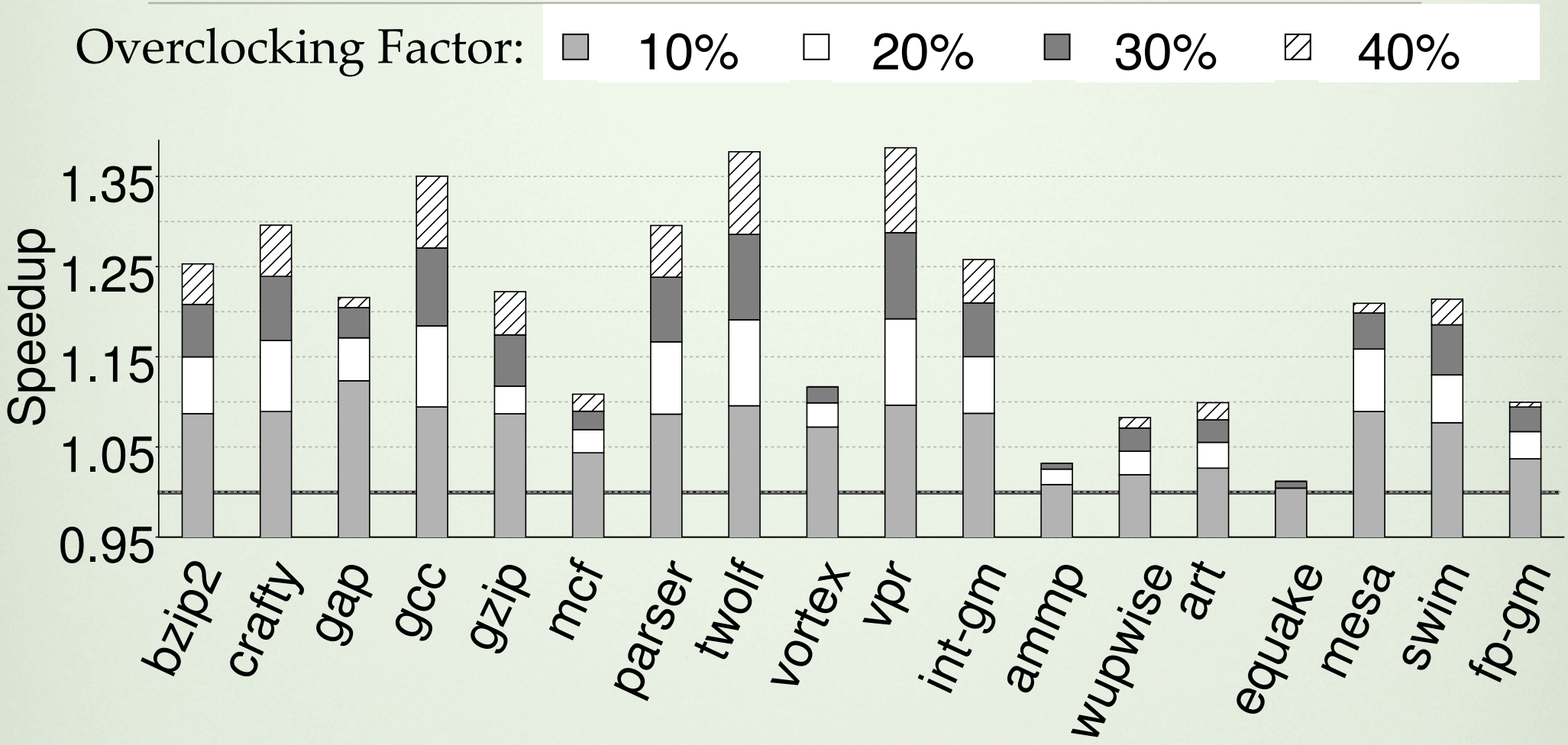
Evaluation

Experiment Setup

- SESC simulator with WATTCH power modeling

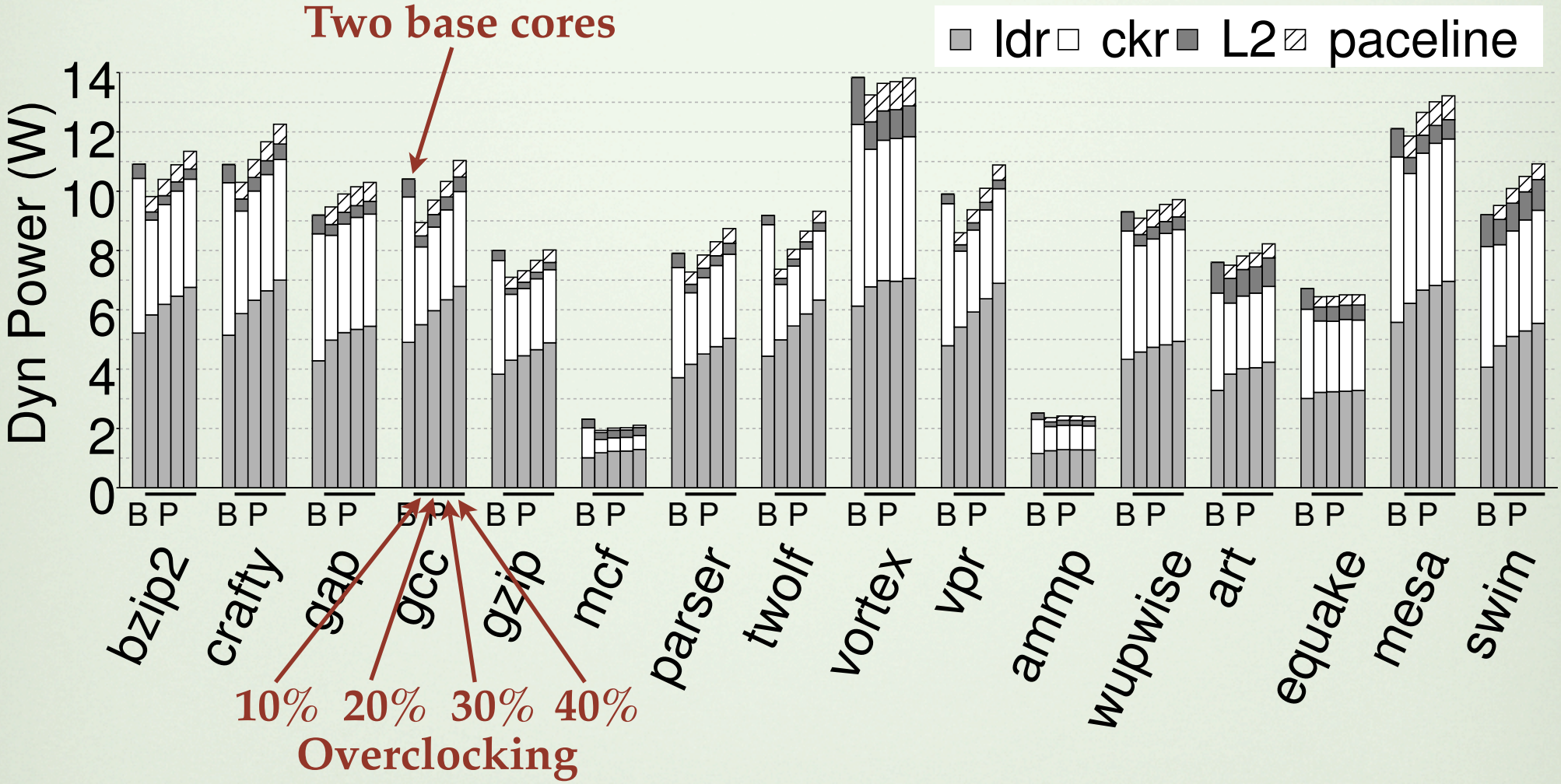
General	16 OoO cores, 32nm, 5 GHz
Core width	6 fetch, 4 issue, 4 retire
ROB size	152
Scheduler size	40 fp, 80 int
LSQ size	54 LD, 46 ST
Branch pred	80Kb local/global tournament, unbounded RAS
L1 I cache	16KB, 2 cyc, 2 port, 2 way
L1 D cache	16KB WT, 2 cyc, 2 port, 4 way
L2 cache	2MB WB, 10 cyc, 1 port, 8 way, shared by two cores, has stride prefetcher
Cache line size	64 bytes
Memory	400 cyc round trip, 10GB/s max

Paceline Speedup



Large speedups for SPECint: 1.21 at 30% overclocking

Paceline Power



Paceline power ≈ Power of two base cores

Conclusions

Paceline: A new approach to improving single-thread performance through replication

- Large performance gains
- Minimal power and thermal impact
- (Optionally) Improved fault tolerance

Paceline

Improving Single-Thread Performance through Core Overclocking

Brian Greskamp and Josep Torrellas
http://iacoma.cs.uiuc.edu/

University of Illinois
at Urbana-Champaign

i-acoma group

Performance Analysis

Obtain from any cycle-accurate simulator

L_i : Instantaneous **Leader speedup** at interval i assuming perfect (infinite-speed) checker

C_i : Instantaneous **Checker speedup** at interval i assuming perfect (infinite-speed) leader

Generate Paceline speedup estimate

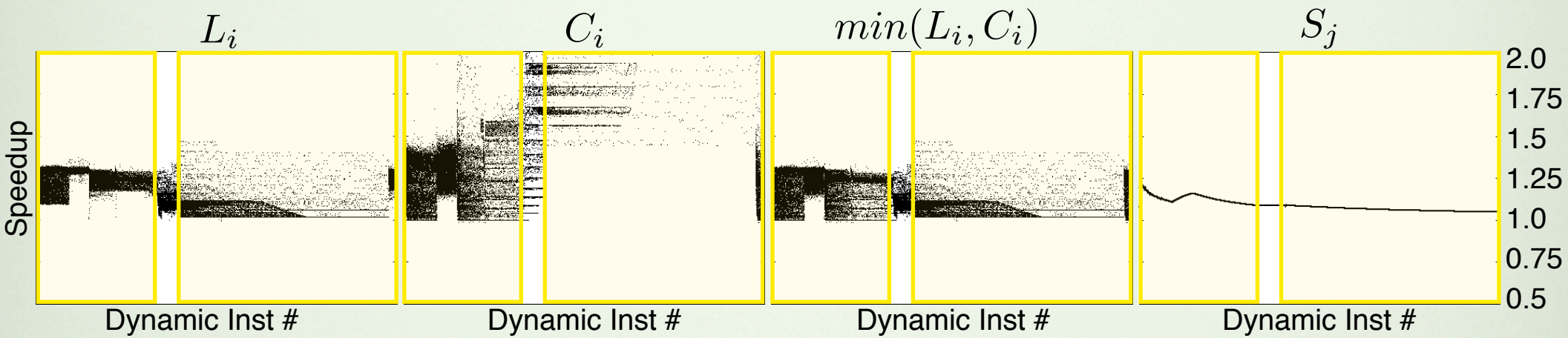
P_i : Instantaneous **Overall speedup** at interval i

S_j : Average **Overall speedup** at interval j

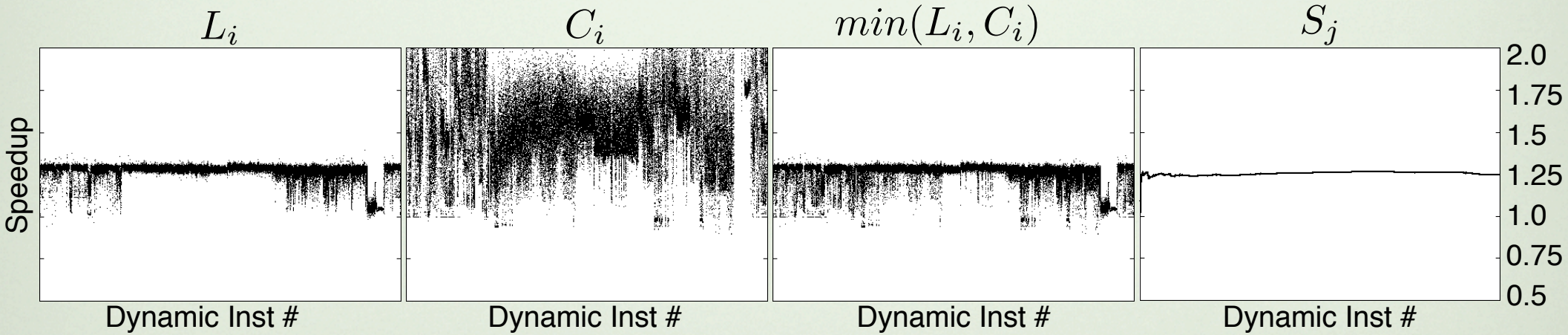
$$P_i = min(L_i, C_i)$$

$$S_j = \frac{j}{\sum_{i=1}^{j} \frac{1}{P_i}}$$

Performance Analysis: Examples



ammp

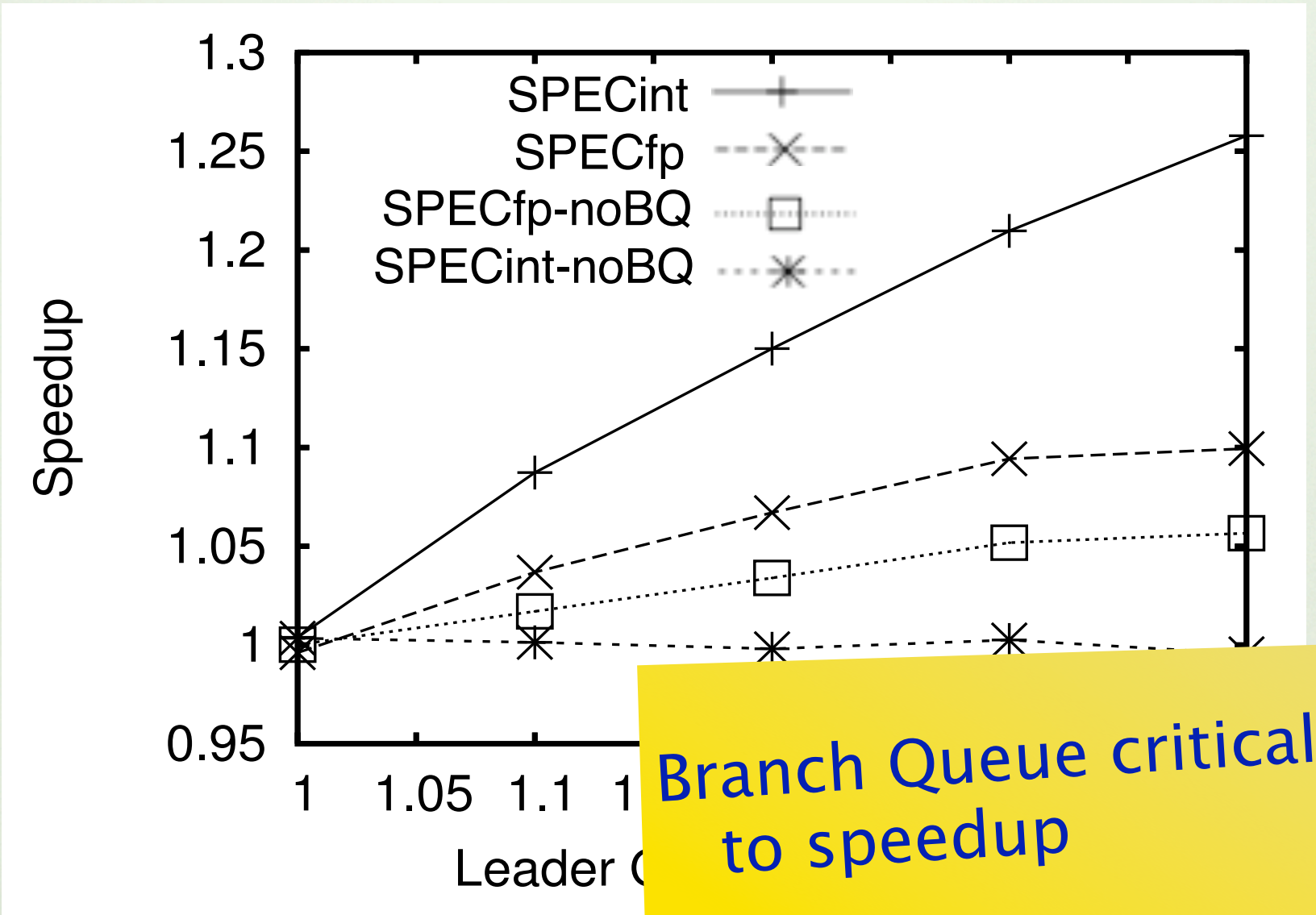


parser

Paceline: Brian Greskamp

i-acoma group

Performance Simulation: Overclocking Factor



Branch Queue critical to speedup

Power Analysis

Obtain from any energy-enabled simulator

E_l : Total **leader energy** (overclocked)

E_c : Total **checker energy** (perfect cache, perfect bpred)

S : Overall Paceline speedup

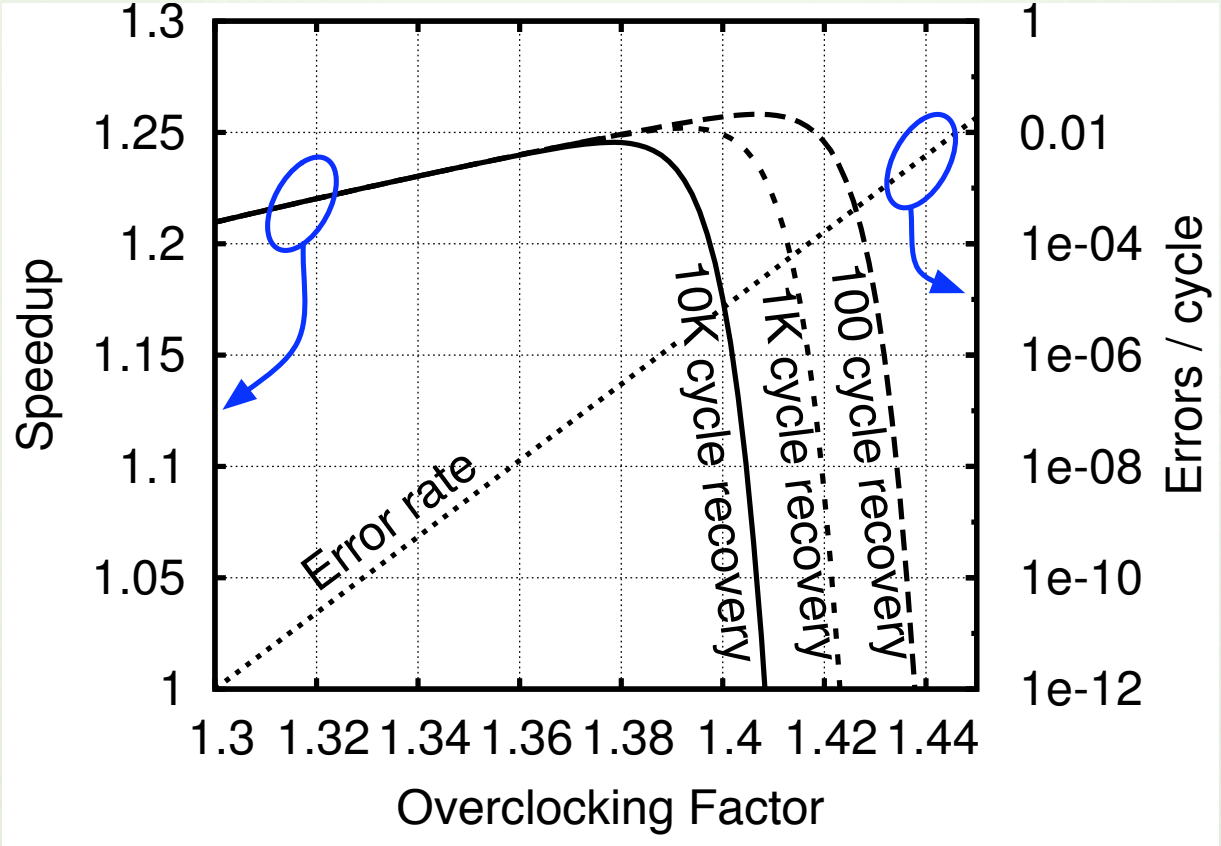
T_b : Execution time on baseline core

Generate Paceline core power estimate

$$P = \frac{S\ (E_l + E_c)}{T_b}$$

- Does not include energy for Paceline structures

Performance Sensitivity



Recovery penalty is not critical to performance

i-acoma group